Package 'attempt'

May 3, 2020

Title Tools for Defensive Programming

Version 0.3.1

Description Tools for defensive programming, inspired by 'purrr' mappers and based on 'rlang'.'attempt' extends and facilitates defensive programming by providing a consistent grammar, and provides a set of easy to use functions for common tests and conditions. 'attempt' only depends on 'rlang', and focuses on speed, so it can be easily integrated in other functions and used in data analysis.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/ColinFay/attempt

LazyData true

Suggests testthat, knitr, rmarkdown, curl

VignetteBuilder knitr

Imports rlang

RoxygenNote 7.1.0

NeedsCompilation no

Author Colin Fay [aut, cre] (<https://orcid.org/0000-0001-7343-1846>)

Maintainer Colin Fay <contact@colinfay.me>

Repository CRAN

Date/Publication 2020-05-03 20:50:02 UTC

R topics documented:

`attempt-package` *attempt package*

Description

Tools for defensive programming in R. 'attempt' extends and facilitates defensive programming by providing a consistent grammar, and provides a set of easy to use functions for common tests and conditions. 'attempt' only depends on `rlang` & `withr`, and focuses on speed, so it can be easily integrated in other functions and used in data analysis.

Author(s)

colin <contact@colinfay.me>

`attempt` *Attempt*

Description

A wrapper around base try that allows you to set a custom message when an error/warning occurs. `attempt` returns the value if there is no error nor message.

Usage

```
attempt(expr, msg = NULL, verbose = FALSE, silent = FALSE)
```

Arguments

`expr`	the expression to be evaluated
`msg`	the message to return if an error occurs
`verbose`	wether or not to return to expression producing the error
`silent`	wether or not the error should be kept under silence

Examples

```
## Not run:
attempt(log("a"), msg = "Nop !")

## End(Not run)
```

`discretly` *discretly*

Description

Prevent a funtion from printing message or warning

Usage

```
discretly(.f)

discreetly(.f)
```

Arguments

`.f`	the function to wrap

Value

an error if any, a warning if any, the result if any

Examples

```
## Not run:
discrete_mat <- discretly(matrix)
discrete_mat(1:3, 2)

## End(Not run)
```

`if_all` *Test for all, any or none*

Description

Test for all, any or none

Usage

```
if_all(.l, .p = isTRUE, .f)

if_any(.l, .p = isTRUE, .f)

if_none(.l, .p = isTRUE, .f)
```

Arguments

.l	the list to test.
.p	the predicate for testing. Defaut is isTRUE.
.f	a mapper or a function run if .p(.x) is TRUE.

Value

If .p(.x) is TRUE, .f() is run.

Examples

```
if_all(1:10, ~ .x < 11, ~ return(letters[1:10]))
if_any(1:10, is.numeric, ~ return(letters[1:10]))
if_none(1:10, is.numeric, ~ return(letters[1:10]))
```

---

if_then *If this, then that*

Description

If this, then that

Usage

```
if_then(.x, .p = isTRUE, .f)

if_not(.x, .p = isTRUE, .f)

if_else(.x, .p = isTRUE, .f, .else)
```

Arguments

.x	the object to test. If NULL (the default), only .p is evaluated.
.p	the predicate for testing. Defaut is isTRUE.
.f	a mapper or a function run if .p(.x) is TRUE
.else	a mapper or a function run if .p(.x) is not TRUE

Value

Depending on wether or not .p(.x) is TRUE, .f() or .else() is run.

Note

If you want these function to return a value, you need to wrap these values into a mapper / a function. E.g, to return a vector, you'll need to write `if_then(1,is.numeric,~ "Yay")`.

Examples

```
a <- if_then(1, is.numeric, ~ "Yay")
a <- if_not(1, is.character, ~ "Yay")
a <- if_else(.x = TRUE, .f = ~ "Yay", .else = ~ "Nay")
```

`is_try_error` *Is the element of class "try-error"?*

Description

Is the element of class "try-error"?

Usage

```
is_try_error(.x)
```

Arguments

`.x`	the object to test

Value

A logical

Examples

```
x <- attempt(log("a"), silent = TRUE)
is_try_error(x)
```

on_error *Add a function to be run on error*

Description

This function behaves as 'on.exit()', but is run on error, and supports mappers.

Usage

```
on_error(f)
```

Arguments

f	a function to call on error

Value

A local error handler.

Examples

```
y <- function(num){
  on_error(~ write( Sys.time(), "error_log.txt", append = TRUE) )
  log(num)
}
```

silently *Silently*

Description

silently returns a new function that will returns an error or a warning if any, or else returns nothing.

Usage

```
silently(.f)
```

Arguments

.f	the function to silence

Value

an error if any, a warning if any. The result is never returned.

Examples

```
## Not run:
silent_log <- silently(log)
silent_log(1)
silent_log("a")

## End(Not run)
```

`silent_attempt` *Silently attempt*

Description

A wrapper around silently and attempt

Usage

```
silent_attempt(...)
```

Arguments

`...`	the expression to evaluate

Value

an error if any, a warning if any.

Examples

```
## Not run:
silent_attempt(warn("nop!"))

## End(Not run)
```

`stop_if` *Warn if*

Description

Friendlier messaging functions.

Usage

```
stop_if(.x, .p = isTRUE, msg = NULL)

stop_if_any(.l, .p = isTRUE, msg = NULL)

stop_if_all(.l, .p = isTRUE, msg = NULL)

stop_if_none(.l, .p = isTRUE, msg = NULL)

stop_if_not(.x, .p = isTRUE, msg = NULL)

warn_if(.x, .p = isTRUE, msg = NULL)

warn_if_any(.l, .p = isTRUE, msg = NULL)

warn_if_all(.l, .p = isTRUE, msg = NULL)

warn_if_none(.l, .p = isTRUE, msg = NULL)

warn_if_not(.x, .p = isTRUE, msg = NULL)

message_if(.x = NULL, .p = isTRUE, msg = NULL)

message_if_any(.l, .p = isTRUE, msg = NULL)

message_if_all(.l, .p = isTRUE, msg = NULL)

message_if_none(.l, .p = isTRUE, msg = NULL)

message_if_not(.x, .p = isTRUE, msg = NULL)
```

Arguments

.x	the element to evaluate. It can be a predicate function (i.e a function returning TRUE).
.p	the predicate with the condition to test on .x or .l. Default is isTRUE.
msg	the message to return. If NULL (default), the built-in message is printed.
.l	the list of elements to evaluate

Examples

```
## Not run:
x <- 12
stop_if(x, ~ .x > 13)
stop_if_not(x, is.character)

a  <- "this is not numeric"
warn_if(a,  is.character )
```

```
warn_if_not(a, is.numeric )
b  <- 20
warn_if(b ,  ~ . > 10 ,
         msg = "Wow, that's a lot of b")
c <- "a"
message_if(c, is.character,
         msg = "You entered a character element")

## End(Not run)
```

---

surely *surely*

Description

Wrap a function in a try

Usage

```
surely(.f)
```

Arguments

.f	the function to wrap

Value

an error if any, a warning if any, the result if any

Examples

```
## Not run:
sure_log <- surely(log)
sure_log(1)
sure_log("a")

## End(Not run)
```

`try_catch` *Try Catch*

Description

Friendlier try catch functions

Usage

```
try_catch(expr, .e = NULL, .w = NULL, .f = NULL)

try_catch_df(expr)

map_try_catch(l, fun, .e = NULL, .w = NULL, .f = NULL)

map_try_catch_df(l, fun)
```

Arguments

`expr`	for simple try catch, the expression to be evaluated
`.e`	a one side formula or a function evaluated when an error occurs
`.w`	a one side formula or a function evaluated when a warning occurs
`.f`	a one side formula or an expression evaluated before returning or exiting
`l`	for map_* function, a list of arguments
`fun`	for map_* function, a function to try with the list `l`

Details

try_catch handles errors and warnings the way you specify. try_catch_df returns a tibble with the call, the error message if any, the warning message if any, and the value of the evaluated expression.

Examples

```
## Not run:
try_catch(log("a"), .e = ~ paste0("There was an error: ", .x))
try_catch(log(1), .f = ~ print("finally"))
try_catch(log(1), .f = function() print("finally"))

## End(Not run)
```

`with_message` *Manipulate messages and warnings*

Description

`with_message` and `with_warning` add a warning or a message to a function. `without_message` and `without_warning` turn the warning and message off.

Usage

```
with_message(.f, msg)

with_warning(.f, msg)

without_message(.f)

without_warning(.f)
```

Arguments

`.f`	the function to wrap
`msg`	the message to print

Value

a function

Examples

```
msg_as_num <- with_message(as.numeric, msg = "Numeric conversion")
warn_as_num <- with_warning(as.numeric, msg = "Numeric conversion")
```

Index