Package 'decorators'

March 22, 2021

Type Package

Title Extend the Behaviour of a Function without Explicitly Modifying it

URL https://tidylab.github.io/decorators/, https://github.com/tidylab/decorators

BugReports https://github.com/tidylab/decorators/issues

Version 0.1.0

Date 2021-03-15

Maintainer Harel Lustiger <tidylab@gmail.com>

Description A decorator is a function that receives a function, extends its behaviour, and returned the altered function. Any caller that uses the decorated function uses the same interface as it were the original, undecorated function. Decorators serve two primary uses: (1) Enhancing the response of a function as it sends data to a second component; (2) Supporting multiple optional behaviours. An example of the first use is a timer decorator that runs a function, outputs its execution time on the console, and returns the original function's result. An example of the second use is input type validation decorator that during running time tests whether the caller has passed input arguments of a particular class. Decorators can reduce execution time, say by memoization, or reduce bugs by adding defensive programming routines.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Language en-GB

Depends R (>= 3.5)

Suggests testthat

Imports purrr

Config/testthat/edition 3

NeedsCompilation no

Author Harel Lustiger [aut, cre] (<https://orcid.org/0000-0003-2953-9598>),
Tidylab [cph, fnd]

Repository CRAN

Date/Publication 2021-03-22 10:20:02 UTC

R topics documented:

NA_x_ 2
time_it 3
validate_arguments 3

Index **6**

`NA_x_` *Not Available / Missing Values*

Description

`NA` is a logical constant of length 1 which contains a missing value indicator. Other built-in constants `NA_integer_`, `NA_real_`, `NA_complex_` and `NA_character_` are missing value place-holder of a particular type.

NA constants have two typical uses:

1. Signifying the user what data type a function expect in its input; and
2. Representing missing values of a specific type.

This package provides additional `NA` types.

Usage

```
NA_Date_

NA_POSIXct_
```

Format

An object of class `Date` of length 1.

An object of class `POSIXct` (inherits from `POSIXt`) of length 1.

Value

`NA` value of a particular type.

Examples

```
class(NA_Date_)
class(NA_POSIXct_)
```

time_it *Measure Execution Time of Functions*

Description

Wrap a function with a timer.

Usage

```
time_it(func, units = "auto", digits = 2)
```

Arguments

func	(function) A function to decorate.
units	(character) Units in which the results are desired, including: "auto", "secs", "mins", "hours", "days", and "weeks". See difftime.
digits	(integer) The number of significant digits to be used. See signif.

Value

(closure) An object that contains the original function bound to the environment of the decorator.

References

- timeit Python module
- Closures in R

Examples

```
Sys.sleep <- time_it(base::Sys.sleep)
Sys.sleep(0.1)
```

validate_arguments *Validate the Type of Input Arguments*

Description

Wrap a function with a input validation.

Usage

```
validate_arguments(func)
```

Arguments

func	(function) A function to decorate.

Details

validate_arguments decorator allows the arguments passed to a function to be parsed and validated using the function's annotations before the function is called.

How It Works:

validate_arguments provides an extremely easy way to apply validation to your code with minimal boilerplate. The original function needs to have key-value pairs in its declaration, where the each value carries its designated class.

When to Use It:

- To protect functions from receiving unexpected types of input arguments.
- In ValueObjects.

Examples: Functions with Built-in NA classes:

Given a Customer ValueObject

```
Customer <- function(given = NA_character_, family = NA_character_)
    return(data.frame(given = given, family = family))
```

When Customer is decorated with validate_arguments

```
Customer <- validate_arguments(Customer)
```

Then passing arguments of any type other then the declared type prompts an informative error.

In the Customer example, both input arguments given and family are declared as character.

```
Customer(given = "Bilbo", family = "Baggins") # Works as both arguments are character
#>   given  family
#> 1 Bilbo Baggins
try(Customer(given = "Bilbo", family = 90201)) # Fails because family is not a character
#> Error in Customer(given = "Bilbo", family = 90201) :
#>   family is of type `numeric` rather than `character`!
```

Value

(closure) An object that contains the original function bound to the environment of the decorator.

Note

The original function must have default values of the designated type.

References

- Similar Python module
- Closures in R

Examples

```
Car <- function(model = NA_character_, hp = NA_real_){
    return(data.frame(model = model, hp = hp))
}

Car <- validate_arguments(Car)
try(Car(model = 555, hp = 120)) # fails because model is numeric rather than character
```

Index

∗ **datasets**
NA_x_, 2
∗ **defensive programming**
validate_arguments, 3
∗ **misc decorators**
time_it, 3

difftime, *3*

NA_Date_ *(*NA_x_*)*, 2
NA_POSIXct_ *(*NA_x_*)*, 2
NA_x_, 2

signif, *3*

time_it, 3

validate_arguments, 3