Package: utilitybeltassertions (via r-universe)

October 22, 2024

Title What the Package Does (One Line, Title Case)

Version 0.0.0.9000

Description What the package does (one paragraph).

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports assertthat, crayon, rlang, utilitybeltfmt (>= 0.0.0.9000)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Remotes selkamand/utilitybeltfmt

Repository https://selkamand.r-universe.dev

RemoteUrl https://github.com/selkamand/utilitybeltassertions

RemoteRef HEAD

RemoteSha bceeb02684ab1c83879bcbde4fe8f17e483f41b8

Contents

`assert_all_values_are_in_set` *Check values are in set*

Description

Check if test_values are one of those described in 'set' vector

Usage

```
assert_all_values_are_in_set(
  test_values,
  acceptable_values,
  name = rlang::caller_arg(test_values)
)
```

Arguments

`test_values`	values to check are equal to one of the values in set (character/numeric)
`acceptable_values`	valid options for elements in test_values (character/numeric)
`name`	name used to describe test values. Used in error message reporting (i.e. 'name' must be one of <acceptable_values>)

Examples

```
set = c("A", "B", "C")
letters = c("A", "B", "B")
assert_all_values_are_in_set(letters, set)
```

`assert_filenames_have_valid_extensions` *Assert file has the expected extension*

Description

Take a filename / vector of filenames and assert that they all end with one of the user-supplied 'valid extensions'

Usage

```
assert_filenames_have_valid_extensions(
  filenames,
  valid_extensions,
  ignore_case = TRUE
)
```

Arguments

filenames	filenames to assert has a valid extension (character)
valid_extensions	all possible valid extensions (character)
ignore_case	does the case (uppercase/lowercase) of the extensions matter? (bool)

Examples

```
# Ensure filename has a "fasta" or 'fa' extension
assert_filenames_have_valid_extensions(
  filename="sequence.fasta",
  valid_extensions = c("fasta", "fa")
)
```

---

assert_files_exist *Do all files exist*

Description

Do all files exist

Usage

```
assert_files_exist(filepaths, supplementary_error_message = "")
```

Arguments

filepaths	filepaths (character)
supplementary_error_message	supplementary error message - e.g. a line describing what functions to run to produce required files (string)

Examples

```
## Not run:
assert_files_exist(
  c("/path/to/file1", "/path/to/file2"),
  supplementary_error_message = "Please run X to produce files you need"
)

## End(Not run)
```

assert_is_whole_number *Check object is a Whole Number*

Description

Checks if object is a whole number (e.g. 1, 5, 5.0, 6.000). Vectors are flagged as NOT whole numbers (intentional behaviour).

Usage

```
assert_is_whole_number(object, msg = "")
```

Arguments

object	Some value you want to assert is a whole number (single scalar value)
msg	Some message to print on failure (appended to the hard-coded message). Will automatically get wrapped in utilitybeltfmt::fmterror (string)

Value

invisible(TRUE) if the object passes the assertion. Throws an error if it does not.

See Also

Other customassertions: `assert_non_empty_string()`, `assert_that_invisible()`

Examples

```
assert_is_whole_number(5)
```

assert_names_include *assert_names_include*

Description

assert_names_include

Usage

```
assert_names_include(object, expected_names, object_name_in_error_message = NA)
```

Arguments

object	an object (usually vector or dataframe) that you want to assert has certain names
expected_names	names you expect the object to have (order doesn't matter) (character vector)
object_name_in_error_message	how to refer to the object in the error message

Examples

```
assert_names_include(mtcars, expected_names = c("mpg", "cyl"))
```

---

assert_non_empty_string

Check object is a non-empty string

Description

Check object is a non-empty string

Usage

```
assert_non_empty_string(object, msg = "")
```

Arguments

object	Some value you want to assert is a non-empty string
msg	Some message to print on failure (appended to the hard-coded message). Will automatically get wrapped in utilitybeltfmt::fmterror (string)

Value

invisible(TRUE) if the object is a non-empty string. Throws an error if it is not.

See Also

Other customassertions: `assert_is_whole_number()`, `assert_that_invisible()`

Examples

```
possiblestring = "Billy"
assert_non_empty_string(possiblestring)
```

assert_program_exists_in_path *Assert a program is in path*

Description

Check if program is available in path Should work on all operating systems

Usage

```
assert_program_exists_in_path(program_names)
```

Arguments

program_names	name/s of program to search for in path (character)

Examples

```
## Not run:
assert_program_exists_in_path(c("grep", "wget"))

## End(Not run)
```

assert_that_invisible *Test Assertion with Invisible Return*

Description

Wraps around asserthat::assertthat::assert_that() but makes return value invisible (can be assigned but will not print when not assigned)

Usage

```
assert_that_invisible(..., env = parent.frame(), msg = NULL)
```

Arguments

...	see ?assertthat::assert_that
env	see ?assertthat::assert_that
msg	see ?assertthat::assert_that

Value

invisible (TRUE) if expression is TRUE. Will error if is FALSE

See Also

Other customassertions: assert_is_whole_number(), assert_non_empty_string()

class_is *class_is*

Description

Check if object has a particular class

Usage

```
class_is(object, tested_class)
```

Arguments

object	object whose class you want to check (object)
tested_class	class (string)

Value

TRUE if object class matches tested_class. FALSE if not.

fmtbold *Colour text*

Description

A collection of functions that take text and return that same text flanked by characters that will lead to its coloration/formatting when printed to terminals using message/cat. Different presets are available: utilitybeltfmt::fmterror, utilitybeltfmt::fmtwarning, utilitybeltfmt::fmtsuccess, utilitybeltfmt::fmtbold.

If greater control is required, use the crayon package

Usage

```
fmtbold(...)
```

Arguments

...	(string/s) Text to colorise. Comma separated strings will be concatenated (no spaces) before colorisation.

Value

(string) Input text flanked by relevant Ansi escape codes

fmterror *Colour text*

Description

A collection of functions that take text and return that same text flanked by characters that will lead to its coloration/formatting when printed to terminals using message/cat. Different presets are available: utilitybeltfmt::fmterror, utilitybeltfmt::fmtwarning, utilitybeltfmt::fmtsuccess, utilitybeltfmt::fmtbold.

If greater control is required, use the crayon package

Usage

```
fmterror(...)
```

Arguments

...	(string/s) Text to colorise. Comma separated strings will be concatenated (no spaces) before colorisation.

Value

(string) Input text flanked by relevant Ansi escape codes

Examples

```
message(utilitybeltfmt::fmterror("This is a warning"))
```

fmtsuccess *Colour text*

Description

A collection of functions that take text and return that same text flanked by characters that will lead to its coloration/formatting when printed to terminals using message/cat. Different presets are available: utilitybeltfmt::fmterror, utilitybeltfmt::fmtwarning, utilitybeltfmt::fmtsuccess, utilitybeltfmt::fmtbold.

If greater control is required, use the crayon package

Usage

```
fmtsuccess(...)
```

Arguments

...	(string/s) Text to colorise. Comma separated strings will be concatenated (no spaces) before colorisation.

Value

(string) Input text flanked by relevant Ansi escape codes

`fmtwarning` *Colour text*

Description

A collection of functions that take text and return that same text flanked by characters that will lead to its coloration/formatting when printed to terminals using message/cat. Different presets are available: utilitybeltfmt::fmterror, utilitybeltfmt::fmtwarning, utilitybeltfmt::fmtsuccess, utilitybeltfmt::fmtbold.

If greater control is required, use the crayon package

Usage

```
fmtwarning(...)
```

Arguments

...	(string/s) Text to colorise. Comma separated strings will be concatenated (no spaces) before colorisation.

Value

(string) Input text flanked by relevant Ansi escape codes

`fun_count_arguments` *How Many Arguments?*

Description

Check how many arguments a function takes.

Usage

```
fun_count_arguments(FUN)
```

Arguments

FUN	a function whose arguments we're going to count (function)

Details

Can be called from inside or outside the function, however if calling from inside a function, the function must be named.

Value

the number of arguments the function takes (numeric)

Examples

```
# Calling from outside a named function
fun = function(a, b, c) { return(a+b-c) }
fun_count_arguments(fun) ##Returns 3

# Calling from outside an anonymous
fun_count_arguments(function(a, b, c) { return(a+b+c) }) ##Returns 3

# Calling from inside a named function
my_function <- function(a, b, c, d, e, f, g) { return(fun_count_arguments(my_function)) }
my_function() # Equals 7
```

---

get_calling_function *Get Calling Function*

Description

Looks through stack trace to identify the function that called the current function.

Usage

```
get_calling_function(n_function_calls_ago = 1, verbose = TRUE)
```

Arguments

n_function_calls_ago	position in the stack trace we're interested in. By default will return the name of the function containing the line: get_calling_function(). Setting to 1 will get the call 1 up in the stack trace.(integer)
verbose	print informative messages

Value

the function call as a string. If function is called from the base environment or n_function_calls_ago is set higher than the number of calls in the stack trace, will return 'base' (string)

Examples

```
wrapper <- function(n_function_calls_ago=1, verbose=TRUE){
  get_calling_function(n_function_calls_ago, verbose)
}

wrapper() # Returns"wrapper()"
```

```
wrapper_wrapper <- function(n_function_calls_ago=1, verbose=TRUE){
  wrapper(n_function_calls_ago, verbose=verbose)
}

wrapper_wrapper() # Returns "wrapper(n_function_calls_ago, verbose = verbose)"
wrapper_wrapper(2) # Returns "wrapper_wrapper(2)"
wrapper_wrapper(100, verbose=FALSE) # Returns "base"
wrapper_wrapper(100, verbose=TRUE) # Returns "base" + message
```

Index