Package 'frankenR'

May 14, 2025

Type Package

Title Lightweight Code Capture and Transformation Toolkit

Version 0.3.1

Description Provides lightweight tools for capturing, manipulating, normalizing, and analyzing R expressions and code blocks. Features include session and script capture, expression atomization, argument modification, selective filtering, and metadata attachment.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Contents

`+.callobj` *Add or modify arguments in a call*

Description

Add or modify arguments in a call

Usage

```
## S3 method for class 'callobj'
x + y
```

Arguments

`x`	A call object.
`y`	A named list (for setting arguments) or a single value (for appending).

`+.code_capture` *Concatenate two code_capture objects*

Description

Adds two ‘code_capture‘ objects together using ‘+‘, combining expressions and metadata.

Usage

```
## S3 method for class 'code_capture'
e1 + e2
```

Arguments

`e1`	First ‘code_capture‘ object.
`e2`	Second ‘code_capture‘ object.

Value

A new combined ‘code_capture‘ object.

-.callobj *Remove arguments from a call*

Description

Remove arguments from a call

Usage

```
## S3 method for class 'callobj'
x - y
```

Arguments

x	A call object.
y	A character vector (names) or numeric vector (positions).

accepts_arg *Check if a function or call accepts a specific argument*

Description

Tests whether a function (or the function of a call) accepts the given argument name.

Usage

```
accepts_arg(func_or_call, arg_name, env = parent.frame())
```

Arguments

func_or_call	A function, function name (character or symbol), or a call expression.
arg_name	The argument name to check for.
env	Environment to search for the function if a name is given (default: parent.frame()).

Value

Logical TRUE/FALSE.

add_arg *Add a new argument to a call*

Description

Appends or inserts a new argument into a call expression.

Usage

```
add_arg(expr, value, name = NULL, position = NULL)
```

Arguments

expr	A expression or callobj
value	The value to add as an argument.
name	Optional name for the new argument.
position	Optional position to insert the argument (1 = first, etc.).

Value

The modified call (or list of calls).

as.list.callobj *Convert a call object to a list*

Description

This method defines ‘as.list()‘ for objects of class ‘"callobj"‘, allowing function call expressions to be treated as lists of their components. It temporarily removes the ‘"call"‘ class before conversion to avoid dispatch issues and ensure correct coercion.

Usage

```
## S3 method for class 'callobj'
as.list(x, ...)
```

Arguments

x	A call object.
...	Unsupported

Value

A list where the first element is the function being called in the expression, followed by its arguments.

atomize_capture *Fully atomize a code_capture object with depth control*

Description

Breaks all complex expressions in a ‘code_capture‘ object into sequences of simpler expressions, with optional control over atomization depth.

Usage

```
atomize_capture(capture, prefix = "tmp", depth = Inf)
```

Arguments

capture	A ‘code_capture‘ object.
prefix	Prefix for naming temporary variables.
depth	Maximum recursion depth for atomization (default: Inf).

Value

A new ‘code_capture‘ object with atomized expressions.

atomize_expr *Fully atomize an expression*

Description

Breaks a call into a sequence of sub-expressions, assigning intermediate results to temporary variables, up to a given depth.

Usage

```
atomize_expr(expr, prefix = "tmp", depth = Inf)
```

Arguments

expr	A call object.
prefix	Prefix for naming temporary variables.
depth	Maximum recursion depth (default: Inf).

Value

A list of calls: assignments followed by the final call.

`atomize_expr_with_counter` *Fully atomize an expression with an external counter and depth control*

Description

Like ‘atomize_expr()‘ but allows a depth limit and initial counter to be passed in.

Usage

```
atomize_expr_with_counter(expr, prefix = "tmp", counter = 1, depth = Inf)
```

Arguments

`expr`	A call object.
`prefix`	Prefix for naming temporary variables.
`counter`	Initial counter value.
`depth`	Maximum recursion depth (default: Inf).

Value

A list containing ‘expressions‘ and updated ‘counter‘.

`atomize_selective_capture` *Selectively atomize parts of a code_capture object*

Description

Decomposes specified function calls into separate expressions across a ‘code_capture‘, respecting a recursion depth.

Usage

```
atomize_selective_capture(capture, fn_names, prefix = "tmp", depth = Inf)
```

Arguments

`capture`	A ‘code_capture‘ object.
`fn_names`	Character vector of functions to atomize.
`prefix`	Prefix for temporary variable names.
`depth`	Maximum recursion depth (default: Inf).

Value

A new ‘code_capture‘ object.

atomize_selective_expr *Selectively atomize parts of an expression*

Description

Decomposes only calls to specified function names into separate expressions, up to a given recursion depth.

Usage

```
atomize_selective_expr(expr, fn_names, prefix = "tmp", depth = Inf)
```

Arguments

expr	A call object.
fn_names	Character vector of function names to atomize.
prefix	Prefix for temporary variable names.
depth	Maximum recursion depth (default: Inf).

Value

A list of expressions, including assignments and the final call.

atomize_selective_expr_with_counter *Selectively atomize an expression with an external counter and depth control*

Description

Like ‘atomize_selective_expr()‘, but accepts and updates an external counter and limits depth.

Usage

```
atomize_selective_expr_with_counter(
  expr,
  fn_names,
  prefix = "tmp",
  counter = 1,
  depth = Inf
)
```

Arguments

expr	A call object.
fn_names	Character vector of functions to atomize.
prefix	Prefix for temporary variable names.
counter	Initial counter value (will be updated).
depth	Maximum recursion depth for selective atomization (default: Inf).

Value

A list with ‘expressions‘ (list of calls) and updated ‘counter‘.

`capture` *General capture interface*

Description

General capture interface

Usage

```
capture(x, script = NULL, envir = parent.frame())
```

Arguments

`x`	A code block ‘‘ or missing
`script`	Optional path to script
`envir`	Environment to associate capture with

Value

A ‘code_capture‘ object or TRUE if session started

`capture_block` *Capture a code block*

Description

Capture a code block

Usage

```
capture_block(expr, realize = FALSE, evaluate = FALSE, envir = parent.frame())
```

Arguments

`expr`	A ‘‘ block
`realize`	Realize arguments (default FALSE)
`evaluate`	Evaluate immediately (default FALSE)
`envir`	Evaluation environment

Value

A ‘code_capture‘ object

capture_script *Capture a script file*

Description

Capture a script file

Usage

```
capture_script(
  path,
  encoding = "UTF-8",
  realize = FALSE,
  evaluate = FALSE,
  envir = parent.frame()
)
```

Arguments

path	Path to script
encoding	File encoding (default UTF-8)
realize	Realize arguments (default FALSE)
evaluate	Evaluate immediately (default FALSE)
envir	Evaluation environment

Value

A ‘code_capture‘ object

change_arg *Change an existing argument in a call*

Description

Changes the value of an existing argument in a call. Throws an error if the argument does not exist.

Usage

```
change_arg(expr, name, new_value)
```

Arguments

expr	A expression or callobj.
name	The argument name to change.
new_value	The new value to assign.

Value

The modified call (or list of calls).

change_func *Change the function in a call*

Description

Replaces the function name of a call while keeping its arguments.

Usage

```
change_func(expr, new_func)
```

Arguments

expr	A expression or callobj
new_func	The new function name (symbol or character).

Value

The modified call (or list of calls).

compress_redundant_versions *Compress redundant immutable variable versions*

Description

If a reassigned variable version is only used once to create another, rewrites later version to use the earlier name and removes redundancy.

Usage

```
compress_redundant_versions(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A new ‘code_capture‘ object with compressed variable names.

diagnose_capture *Run diagnostics on a capture object*

Description

Returns a structured list of diagnostic results indicating stochasticity, side effects, non-standard evaluation, and more.

Usage

```
diagnose_capture(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

An object of class ‘"code_diagnosis"‘.

duplicate_line *Duplicate an expression in a code_capture object*

Description

Inserts a duplicate of the expression at index ‘i‘ directly after the original.

Usage

```
duplicate_line(capture, i)
```

Arguments

capture	A ‘code_capture‘ object.
i	Integer index of the expression to duplicate.

Value

A modified ‘code_capture‘ object with duplicated expression.

end_capture *Finalize session capture*

Description

Finalize session capture

Usage

```
end_capture(realize = FALSE, envir = parent.frame())
```

Arguments

realize	Realize arguments (default FALSE)
envir	Evaluation environment

Value

A list of ‘callobj‘ objects

export_capture *Export captured code to a script file*

Description

Writes the expressions from a ‘code_capture‘ object into a script file.

Usage

```
export_capture(
  capture,
  path,
  overwrite = FALSE,
  meta = c("none", "code", "comments")
)
```

Arguments

capture	A ‘code_capture‘ object.
path	The file path to write to.
overwrite	Logical; overwrite existing file? (default: FALSE).
meta	Character; one of "none", "code", or "comments" (default: "none").

Details

The ‘meta‘ argument controls how metadata is handled during export:

- `"none"`: Metadata is ignored. Only the captured expressions are written.
- `"comments"`: Metadata is exported as comments (`# key: value`) immediately before each associated expression.
- `"code"`: Metadata is exported as actual `meta(...)` function calls immediately before each associated expression.

Expressions are exported exactly as captured, without wrapping, indenting, or additional blank lines.

Value

Invisibly TRUE on success.

`filter_by_function` *Filter captured expressions by function name*

Description

Selects only those expressions where the top-level function matches one of the specified names.

Usage

```
filter_by_function(capture, fn_names)
```

Arguments

`capture`	A ‘code_capture‘ object.
`fn_names`	A character vector of function names to keep.

Value

A filtered ‘code_capture‘ object.

`filter_by_predicate` *Filter expressions using a custom predicate*

Description

Selects expressions for which the given predicate function returns TRUE.

Usage

```
filter_by_predicate(capture, predicate)
```

Arguments

capture	A ‘code_capture‘ object.
predicate	A function taking a call and returning TRUE or FALSE.

Value

A filtered ‘code_capture‘ object.

---

get_all_arguments *Get all arguments from all expressions in a capture*

Description

Extracts arguments for each call in a ‘code_capture‘ object.

Usage

```
get_all_arguments(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A list of argument lists.

---

get_arg *Get an argument from a call*

Description

Retrieves the value of a named argument in a function call.

Usage

```
get_arg(expr, name)
```

Arguments

expr	A expression or callobj.
name	The argument name to retrieve.

Value

The argument value, or NULL if not found.

get_arguments *Get the arguments of a call*

Description

Returns all arguments of a call, excluding the function.

Returns a list of arguments from a call expression.

Usage

```
get_arguments(expr)

get_arguments(expr)
```

Arguments

expr	A call object.

Value

A list of argument expressions.

A list of arguments, or NULL if not a call.

get_argument_names *Get the names of arguments in a call*

Description

Returns the argument names (or empty strings for unnamed arguments).

Usage

```
get_argument_names(expr)
```

Arguments

expr	A call object.

Value

A character vector of argument names.

get_expressions *Access expressions from a capture object*

Description

Access expressions from a capture object

Usage

```
get_expressions(capture)
```

Arguments

capture	A ‘code_capture‘ object

Value

A list of expressions

get_expr_meta *Get expression and metadata pair at a given index*

Description

Retrieves the combined expression and metadata list from a ‘code_capture‘ object.

Usage

```
get_expr_meta(capture, i)
```

Arguments

capture	A ‘code_capture‘ object.
i	Integer index to retrieve.

Value

A list containing ‘expr‘ and ‘meta‘.

get_expr_text *Get the deparsed text of expressions in a capture*

Description

Converts all expressions in a ‘code_capture‘ object to character strings.

Usage

```
get_expr_text(capture, collapse = "\n")
```

Arguments

capture	A ‘code_capture‘ object.
collapse	String used to collapse multi-line expressions.

Value

A character vector with one element per expression.

get_function *Get the function called in an expression*

Description

Extracts the function from a call expression.

Usage

```
get_function(expr)
```

Arguments

expr	A call object.

Value

The function name or call (symbol or call).

get_function_name *Get the function name from a call*

Description

Extracts the function name (symbol) from a call expression.

Usage

```
get_function_name(x)
```

Arguments

x	An expression or call object.

Value

The function name as a symbol, or NULL if not a call.

get_inputs *Identify inputs used before they are assigned*

Description

Examines a code_capture object and returns inputs that are used before being defined.

Usage

```
get_inputs(capture)
```

Arguments

capture	A code_capture object.

Value

A named list mapping each input symbol to a list with its first usage index and whether it was used before definition.

get_lhs *Get the left-hand side of an assignment*

Description

Extracts the variable being assigned to in an assignment call.

Usage

```
get_lhs(expr)
```

Arguments

expr	A call object.

Value

A character vector of the LHS variable name.

get_metadata *Access metadata from a capture object*

Description

Access metadata from a capture object

Usage

```
get_metadata(capture)
```

Arguments

capture	A ‘code_capture‘ object

Value

A list of metadata

get_operator *Get the operator of a call*

Description

Returns the operator as character, or NULL if not an operator.

Usage

```
get_operator(expr)
```

Arguments

expr	A call or callobj.

Value

Character name of operator, or NULL.

get_rhs *Get the right-hand side of an assignment*

Description

Extracts the value or expression assigned in an assignment call.

Usage

```
get_rhs(expr)
```

Arguments

expr	A call object.

Value

The RHS expression.

get_symbols *Extract symbols used in a call, separated by LHS and RHS*

Description

Returns a list of symbols and their line numbers for both the left-hand side (LHS) and right-hand side (RHS) of an expression. Works for assignment calls.

Usage

```
get_symbols(expr, parse_data = NULL)
```

Arguments

expr	An R call object (e.g., as captured from parse()).
parse_data	Optional result from getParseData() on the full expression source.

Value

A list with lhs and rhs components, each containing symbols and line numbers.

get_top_function_names *Get top-level function names from expressions*

Description

Extracts the function name from each expression in a ‘code_capture‘ object.

Usage

```
get_top_function_names(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A character vector of function names (or NA for non-calls).

has_arg *Check if a call has a specific argument*

Description

Tests whether a named argument exists in a function call.

Usage

```
has_arg(expr, name)
```

Arguments

expr	A expression or callobj
name	The argument name to check for.

Value

Logical TRUE/FALSE.

has_operator *Check if a call has an operator as function*

Description

Check if a call has an operator as function

Usage

```
has_operator(expr)
```

Arguments

expr	A call or callobj.

Value

TRUE if the function name is a base operator.

immutabilize_capture *Make variables immutable by renaming reassignments*

Description

Rewrites a ‘code_capture‘ so variables are never reassigned. Each assignment to the same variable creates a new name (e.g., ‘x._1‘, ‘x._2‘, ...), and all later uses are updated to use the latest version.

Usage

```
immutabilize_capture(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A new ‘code_capture‘ object with immutable assignments.

isolate_capture *Isolate code by realizing inputs*

Description

Evaluates only the inputs used before they are defined, and prepends assignments so the capture becomes self-contained.

Usage

```
isolate_capture(capture, envir = parent.frame())
```

Arguments

capture	A ‘code_capture‘ object.
envir	An environment in which to evaluate inputs (default: ‘parent.frame()‘).

Value

A new ‘code_capture‘ object with input assignments prepended. Issues warnings for any inputs not found in the environment

is_assignment *Check if an expression is an assignment*

Description

Determines whether an expression is an assignment call (‘<-‘, ‘=‘, or ‘«-‘).

Usage

```
is_assignment(expr)
```

Arguments

expr	A call or callobj.

Value

TRUE if expression is assignment, FALSE otherwise.

is_call_or_list *Check if an object is a call or a list of calls*

Description

Determines whether the input is a single function call or a list entirely composed of calls.

Usage

```
is_call_or_list(x)
```

Arguments

x	An object to check.

Value

Logical TRUE/FALSE.

is_compound *Check if an expression is a compound (nested) call*

Description

Returns TRUE if any arguments of the call are themselves calls.

Usage

```
is_compound(expr)
```

Arguments

expr	A call or callobj.

Value

Logical TRUE/FALSE.

is_function *Check if an expression is a function call*

Description

Check if an expression is a function call

Usage

```
is_function(expr)
```

Arguments

expr	An expression.

Value

TRUE if expr is a call, FALSE otherwise.

length.code_capture *Length of a code_capture object*

Description

Length of a code_capture object

Usage

```
## S3 method for class 'code_capture'
length(x)
```

Arguments

x	‘code_capture‘ object.

Value

The number of expressions in the capture

meta *Metadata function for attaching metadata*

Description

‘meta()‘ is a placeholder function used inside capture sessions to attach metadata to previous expressions. It does nothing at runtime.

Usage

```
meta(...)
```

Arguments

...	Named arguments representing metadata.

Value

Nothing. Intended for side effects only in capture processing.

normalize_call *Normalize a single call by naming unnamed arguments*

Description

Attempts to add names to unnamed arguments in a call based on the function's formals.

Usage

```
normalize_call(expr, env = parent.frame(), partial_match = TRUE, strict = TRUE)
```

Arguments

expr	A call object or callobj.
env	Environment to find the function (default: 'parent.frame()').
partial_match	Logical; whether to allow partial matching of argument names (default: TRUE).
strict	Logical; whether to forcibly assign argument names even for positional matches (default: TRUE).

Value

The modified call or callobj (same type as input).

normalize_capture *Normalize a code_capture object*

Description

Normalizes expressions inside a code_capture object by naming unnamed arguments based on the corresponding function's formals.

Usage

```
normalize_capture(capture, env = parent.frame(), partial_match = TRUE)
```

Arguments

capture	A code_capture object.
env	Environment for function lookup (default: parent.frame()).
partial_match	Logical; allow partial matching of argument names (default: TRUE).

Value

A new normalized code_capture object.

print.code_capture *Print method for code_capture objects*

Description

Print method for code_capture objects

Usage

```
## S3 method for class 'code_capture'
print(x, with_meta = TRUE, ...)
```

Arguments

x	A ‘code_capture‘ object
with_meta	Boolean; Should metadata be printed
...	Ignored

print.code_diagnosis *Print method for code diagnosis*

Description

Displays the results of a ‘code_diagnosis‘ object in a readable format.

Usage

```
## S3 method for class 'code_diagnosis'
print(x, ...)
```

Arguments

x	An object of class ‘"code_diagnosis"‘.
...	Ignored.

Value

Invisibly returns the diagnosis object.

realize_args *Realize (evaluate) the arguments of a call*

Description

Evaluates all arguments inside a function call, leaving the function name unchanged. For assignment calls, only the RHS is evaluated; the LHS is preserved as a symbol. If an evaluation fails, the original unevaluated expression is kept.

Usage

```
realize_args(expr, envir = parent.frame())
```

Arguments

expr	A call or list of calls.
envir	Environment for evaluation (default: ‘parent.frame()‘).

Value

A call with realized arguments, or a list of realized calls.

realize_capture *Realize arguments across a code_capture object*

Description

Evaluates the arguments of all expressions in a ‘code_capture‘ object.

Usage

```
realize_capture(capture, envir = parent.frame())
```

Arguments

capture	A ‘code_capture‘ object.
envir	Environment for evaluation (default: ‘parent.frame()‘).

Value

A new ‘code_capture‘ object with realized expressions.

remove_arg *Remove an argument from a call*

Description

Deletes a named argument from a call expression.

Usage

```
remove_arg(expr, name)
```

Arguments

expr	A expression or callobj.
name	The argument name to remove.

Value

The modified call (or list of calls).

remove_redundant_assignments *Remove redundant self-assignments*

Description

Removes expressions where a variable is assigned to itself (e.g., ‘x <- x‘).

Usage

```
remove_redundant_assignments(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A modified ‘code_capture‘ object with redundant assignments removed.

replace_function *Replace function in all calls inside an expression*

Description

Recursively replaces any function matching old_func with new_func.

Usage

```
replace_function(expr, old_func, new_func)
```

Arguments

expr	A call, callobj, or expression.
old_func	Character name of the function to replace.
new_func	Character or symbol of the replacement function.

Value

The modified expression (same type as input).

replace_operator *Replace operator in a call expression*

Description

Recursively replaces any operator matching old_op with new_op.

Usage

```
replace_operator(expr, old_op, new_op)
```

Arguments

expr	A call, callobj, or expression.
old_op	Character name of the operator to replace.
new_op	Character or symbol of the new operator.

Value

The modified expression (same type as input).

replace_variable *Replace variable name(s) in an expression*

Description

Recursively substitutes symbol names based on a mapping.

Usage

```
replace_variable(expr, mapping)
```

Arguments

expr	A call, callobj, or expression.
mapping	A named list or named character vector (old names = names, new names = values).

Value

The modified expression (same type as input).

rerun_capture *Rerun captured code*

Description

Evaluates expressions from a ‘code_capture‘ object.

Usage

```
rerun_capture(
  capture,
  envir = parent.frame(),
  verbose = FALSE,
  stop_on_error = TRUE,
  collect_results = TRUE,
  new_env = FALSE
)
```

Arguments

capture	A ‘code_capture‘ object.
envir	The environment to evaluate in (default: parent.frame()).
verbose	Logical; print each expression before evaluation (default: FALSE).
stop_on_error	Logical; stop immediately if an error occurs (default: TRUE).
collect_results	Logical; collect and return results in a list (default: TRUE).
new_env	Logical; if TRUE, create a new clean environment (default: FALSE).

Value

A list of results (if collect_results = TRUE), otherwise invisibly TRUE.

---

set_all_metadata *Replace all metadata in a code_capture object*

Description

Updates the metadata for a specific expression in a ‘code_capture‘ object.

Usage

```
set_all_metadata(capture, meta)
```

Arguments

capture	A ‘code_capture‘ object.
meta	A list of named lists containing metadata.

Value

A modified ‘code_capture‘ object.

---

set_arg *Set or add an argument in a call*

Description

Sets an argument to a new value in a call expression (or adds it if not present).

Usage

```
set_arg(expr, name, value)
```

Arguments

expr	A expression or callobj.
name	The argument name.
value	The new value for the argument.

Value

The modified call (or list of calls).

set_expression *Replace an expression in a code_capture object*

Description

Updates the expression at a specific index within a ‘code_capture‘ object.

Usage

```
set_expression(capture, i, expr)
```

Arguments

capture	A ‘code_capture‘ object.
i	Integer index of the expression to replace.
expr	A call object representing the new expression.

Value

A modified ‘code_capture‘ object.

set_expressions *Replace all expressions in a code_capture object*

Description

Replace all expressions in a code_capture object

Usage

```
set_expressions(capture, exprs)
```

Arguments

capture	A ‘code_capture‘ object.
exprs	A list of call objects

Value

A modified ‘code_capture‘ object.

set_function *Set the function of a call expression*

Description

Replaces the function being called.

Usage

```
set_function(expr, fn)
```

Arguments

expr	A call or callobj.
fn	A symbol, call, or character to set as the new function.

Value

The modified call or callobj (same type as input).

set_lhs *Set the left-hand side of an assignment*

Description

Replaces the variable name on the LHS of an assignment call.

Usage

```
set_lhs(expr, lhs)
```

Arguments

expr	A call or callobj.
lhs	A symbol or character name for the new LHS variable.

Value

The modified call or callobj (same type as input).

set_metadata *Replace metadata in a code_capture object*

Description

Updates the metadata for a specific expression in a ‘code_capture‘ object.

Usage

```
set_metadata(capture, i, meta)
```

Arguments

capture	A ‘code_capture‘ object.
i	Integer index of the metadata entry to replace.
meta	A named list containing metadata.

Value

A modified ‘code_capture‘ object.

set_rhs *Set the right-hand side of an assignment*

Description

Replaces the RHS value or expression in an assignment call.

Usage

```
set_rhs(expr, rhs)
```

Arguments

expr	A call or callobj.
rhs	An expression for the new RHS.

Value

The modified call or callobj (same type as input).

simplify_capture *Simplify a capture by removing unused constant assignments*

Description

Removes constant assignments to variables that are never used later.

Usage

```
simplify_capture(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A simplified ‘code_capture‘ object.

sort_capture *Sort expressions in a code_capture object*

Description

Sorts constant assignments and function definitions to the top, preserving relative order and associated metadata.

Usage

```
sort_capture(capture)
```

Arguments

capture	A ‘code_capture‘ object.

Value

A new ‘code_capture‘ object with sorted expressions.

standardize_assignments *Standardize assignment operators to '<-'*

Description

Rewrites any '=' assignments to use '<-' for consistency.

Usage

```
standardize_assignments(capture)
```

Arguments

capture	A 'code_capture' object.

Value

A modified 'code_capture' object.

start_capture *Start capturing top-level expressions*

Description

Start capturing top-level expressions

Usage

```
start_capture(clear = TRUE, envir = parent.frame(), nframe = sys.nframe())
```

Arguments

clear	Whether to clear previous buffer
envir	Environment to associate capture with
nframe	Internal safeguard

Value

Invisibly TRUE

substitute_symbols *Substitute symbols in an expression*

Description

Recursively walks an expression and replaces any symbols found in ‘rename_map‘.

Usage

```
substitute_symbols(expr, rename_map)
```

Arguments

expr	A call or symbol object (or callobj).
rename_map	A named list or character vector mapping old names to new names.

Value

The transformed expression (same type as input).

unwrap_expr *Unwrap the outer function call of an expression*

Description

Removes the top-level function call and returns the first argument inside.

Usage

```
unwrap_expr(expr)
```

Arguments

expr	A expression or callobj

Value

The unwrapped expression(s).

verify_capture *Verify output of a capture against an environment*

Description

Compares the values of assigned variables from a ‘code_capture‘ object against those in a given environment, checking for consistency and reproducibility.

Usage

```
verify_capture(capture, envir = parent.frame(), details = FALSE)
```

Arguments

capture	A ‘code_capture‘ object to verify.
envir	The environment considered to contain the correct reference values. Defaults to the calling environment.
details	Logical; if TRUE, returns a list of mismatches and matches by category. If FALSE, returns a simple TRUE/FALSE.

Details

The function re-executes the capture in a copy of the given environment and compares the results of all variables assigned during the capture. If ‘details = TRUE‘, it returns a list categorizing matched, mismatched, and missing variables.

Value

Either a logical (TRUE/FALSE) or a list with fields:

matches Variables that matched exactly.

value_mismatches Variables that exist in both but differ in value.

missing_in_capture_result Variables expected but not produced by the capture.

missing_in_reference Variables produced by the capture but missing in the reference environment.

which_expr_matches *Identify which expressions match a function + argument pattern*

Description

Checks a list of expressions and returns a logical vector indicating which expressions match any function name in ‘fn_map‘, optionally restricted to specific argument names.

Usage

```
which_expr_matches(exprs, fn_map)
```

Arguments

exprs	A list of expressions (calls or callobj objects).
fn_map	A named list where names are function names and values are character vectors of argument names (empty vector = match any call to the function).

Value

A logical vector, TRUE for matches, FALSE otherwise.

---

wrap_expr *Wrap an expression inside a function*

Description

Creates a new call by wrapping the given expression inside another function call.

Usage

```
wrap_expr(expr, wrapper_fn)
```

Arguments

expr	A expression or callobj
wrapper_fn	A function name (symbol or character) to wrap with.

Value

A wrapped call (or list of calls).

[.callobj *Extract an argument by name or position*

Description

Extract an argument by name or position

Usage

```
## S3 method for class 'callobj'
x[i, ...]
```

Arguments

x	A call object.
i	A character (name) or numeric (position).
...	Ignored. Included for method consistency.

[.code_capture *Subset a code_capture object*

Description

Provides subsetting (‘[‘) for ‘code_capture‘ while preserving the class and metadata.

Usage

```
## S3 method for class 'code_capture'
x[i, ...]
```

Arguments

x	A ‘code_capture‘ object.
i	Subset indices.
...	Ignored.

Value

A subsetted ‘code_capture‘ object.

[<-.code_capture *Replace expressions inside a code_capture*

Description

Provides assignment (‘[<-‘) for replacing elements of a ‘code_capture‘ object.

Usage

```
## S3 replacement method for class 'code_capture'
x[i] <- value
```

Arguments

x	A ‘code_capture‘ object.
i	Index to replace.
value	New value(s) (must be a call or list of calls).

Value

The modified ‘code_capture‘ object.

[[.code_capture *Extract an expression from a code_capture object*

Description

Allows using '[[' to access a specific captured expression.

Usage

```
## S3 method for class 'code_capture'
x[[i, ...]]
```

Arguments

x	A 'code_capture' object.
i	A single index.
...	Ignored.

Value

A single expression (call object).

[[<-.code_capture *Replace an expression inside a code_capture object*

Description

Allows using '[[<-' to assign a new expression at a specific index.

Usage

```
## S3 replacement method for class 'code_capture'
x[[i]] <- value
```

Arguments

x	A 'code_capture' object.
i	Index to replace.
value	A call object (expression) to assign.

Value

The modified 'code_capture' object.

Index