Package: qlcheckr (via r-universe)

May 24, 2026

Title Helpful Code Checking for Quarto Live

Version 0.1.5

Description Provides a framework for interactive code checking in Quarto Live. The package enables users to define logical tests for code correctness, delivers custom feedback messages, and includes debugging capabilities to display evaluation details and error hints. Designed to enhance teaching, learning, and automated code assessment workflows.

License MIT + file LICENSE

Imports stats, utils

Suggests evaluate, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Repository https://startr-academy.r-universe.dev

Date/Publication 2025-07-28 11:45:32 UTC

RemoteUrl https://github.com/startr-academy/qlcheckr

RemoteRef HEAD

RemoteSha e2a21d32e18b0f30198565dd5b228f0490ee7d8a

Contents

apply_checks 2
exists_in 3
outputs-extraction 4
search_ast 5

Index **7**

`apply_checks` *Apply Code Checks for Quarto Live with Feedback and Debugging*

Description

This function evaluates user code against a series of named tests, providing feedback based on the test outcomes. It integrates with Quarto Live to facilitate interactive code checking.

Usage

```
apply_checks(
  ...,
  .msg_correct = "That's correct! Well done.",
  .msg_incorrect = "That's incorrect! Please try again...",
  .debug = FALSE
)
```

Arguments

`...`	Named logical expressions representing the tests to be evaluated. Each test's name should provide a hint or message to display when the test fails. A test passes if it evaluates to `FALSE` and fails if it evaluates to `TRUE`.
`.msg_correct`	Character. The message to display when all tests pass. Default is `"That's correct! Well done."`.
`.msg_incorrect`	Character. The message to display when any test fails. Default is `"That's incorrect! Please try again..."`.
`.debug`	Logical. If `TRUE`, prints detailed debug information, including the state of all tests and additional internal evaluation details. Default is `FALSE`.

Value

Returns `.msg_correct` if all tests pass (`FALSE`), or `.msg_incorrect` followed by the failure hints for tests that failed (`TRUE`). When `.debug` is `TRUE`, additional debug information is printed.

Examples

```
# Example usage:
apply_checks(
  "Your math doesn't work correctly." = 1 + 1 != 2,
  "Your logic is flawed." = 3 * 3 != 9
)
# Returns: "That's correct! Well done."

apply_checks(
  "Your math doesn't work correctly." = 1 + 1 != 2,
  "Your logic is flawed." = 3 * 3 != 8
)
```

```
# Returns:
# "That's incorrect! Please try again...
# - Your logic is flawed."

apply_checks(
  "Your math doesn't work correctly." = 1 + 1 != 2,
  .debug = TRUE
)
# Debug output includes detailed evaluation results and failure messages.
```

`exists_in` *Check for Existence in a List Based on a Condition*

Description

Evaluates whether elements in a list satisfy a given condition and combines the results using a specified logical function (e.g., `any` or `all`).

Usage

```
exists_in(.x, .f, ..., .require = any)
```

Arguments

`.x`	A list or vector to be evaluated.
`.f`	A predicate function or a formula. If a formula is provided, it will be converted to a function. The formula should use `.` to refer to elements of `.x`.
`...`	Additional arguments passed to `.f`.
`.require`	A logical function to combine the results of applying `.f` to each element of `.x`. Default is `any()`, which returns `TRUE` if any elements satisfy the condition.

Value

A logical value. `TRUE` if `.require` evaluates to `TRUE` for the results of applying `.f` to `.x`, otherwise `FALSE`.

Examples

```
# Example with a function
exists_in(list(1, 2, 3), function(x) x > 2)
# Returns TRUE (since 3 > 2)

# Example with a formula
exists_in(list(1, 2, 3), ~ . > 2)
# Returns TRUE (since 3 > 2)

# Example requiring all elements to satisfy the condition
```

```
exists_in(list(1, 2, 3), ~ . > 0, .require = all)
# Returns TRUE (all elements are > 0)

# Example with additional arguments
exists_in(list("apple", "banana", "cherry"), grepl, pattern = "a")
# Returns TRUE (some elements contain "a")
```

`outputs-extraction` *Extract Common Output Types from an Evaluate Object*

Description

These functions facilitate the extraction of specific output types, such as results, errors, warnings, or messages, from an `evaluate` object. The outputs are filtered based on their class and a specified field, making it easier to analyse the results of code evaluation in Quarto Live exercise grading.

Usage

```
ql_outputs(class, field)

ql_results()

ql_errors()

ql_warnings()

ql_messages()

ql_src()

ql_ast()
```

Arguments

`class`	Character. The class of objects to extract from the `evaluate` output.
`field`	Character. The field within the object to extract (e.g., `"value"` or `"message"`).

Details

The functions operate on an `.evaluate_result` object, which stores outputs from code execution (e.g., in Quarto Live exercises). They are designed to extract specific elements such as:

- **Results**: Extracts the `value` field from objects of class `"results"`.
- **Errors**: Extracts the `message` field from objects of class `"error"`.
- **Warnings**: Extracts the `message` field from objects of class `"warning"`.
- **Messages**: Extracts the `message` field from objects of class `"message"`.

Value

A list containing the extracted fields from the specified class of outputs.

Examples

```
.evaluate_result <- evaluate::evaluate(
'print(rnorm(10))
sample$x
log(-1)
message("Hello world")', output_handler = ql_output_handler)

# Extract results
ql_results()

# Extract errors
ql_errors()

# Extract warnings
ql_warnings()

# Extract messages
ql_messages()
```

---

search_ast *Search Abstract Syntax Tree (AST) for Specific Patterns*

Description

Analyses parsed user code to identify the presence of specific patterns in the Abstract Syntax Tree (AST), such as the usage of functions, arguments, or expressions.

Usage

```
search_ast(.code, .fn = NULL, ..., .expr = NULL)
```

Arguments

.code	A parsed R expression or a call object representing the user code to analyse. Typically obtained using parse() or similar methods.
.fn	Character or NULL. The name of the function to search for in the code. If NULL, the function usage is not checked.
...	Additional arguments to search for in function calls. These may include named or unnamed arguments.
.expr	A quoted or parsed R expression to search for within the code. If NULL, expressions are not checked. Using .expr has priority over .fn and ..., if both conditions are specified only .expr will be used.

Details

The function operates on the AST representation of the provided code, enabling precise pattern matching. Use `.fn` to check for the presence of a specific function, `...` to search for arguments, and `.expr` for custom expression matching. Any combination of these can be used to tailor the search criteria.

Value

A logical value indicating whether the specified patterns were found in the provided code (`TRUE`) or not (`FALSE`).

Examples

```
# Example: Search for a specific function name
search_ast(quote(mean(x)), .fn = mean)

# Example: Search for a specific argument
search_ast(quote(mean(x, na.rm = TRUE)), na.rm = TRUE)
search_ast(quote(mean(x, na.rm = TRUE)), na.rm = FALSE)

# Example: Search for an expression
search_ast(quote(mean(x + y)), .expr = x + y)
search_ast(quote(mean(x + y)), .expr = mean(x + y))
search_ast(quote(mean(x + y)), .expr = log(x + y))
```

Index

any(), *3*
apply_checks, 2

errors *(*outputs-extraction*)*, 4
exists_in, 3

messages *(*outputs-extraction*)*, 4

outputs *(*outputs-extraction*)*, 4
outputs-extraction, 4

parse(), *5*

ql_ast *(*outputs-extraction*)*, 4
ql_errors *(*outputs-extraction*)*, 4
ql_messages *(*outputs-extraction*)*, 4
ql_outputs *(*outputs-extraction*)*, 4
ql_results *(*outputs-extraction*)*, 4
ql_src *(*outputs-extraction*)*, 4
ql_warnings *(*outputs-extraction*)*, 4

results *(*outputs-extraction*)*, 4

search_ast, 5

warnings *(*outputs-extraction*)*, 4