Package 'shinyvalidate'

October 4, 2023

Title Input Validation for Shiny Apps

Version 0.1.3

Description Improves the user experience of Shiny apps by helping to provide feedback when required inputs are missing, or input values are not valid.

License MIT + file LICENSE

URL https://rstudio.github.io/shinyvalidate/, https://github.com/rstudio/shinyvalidate

BugReports https://github.com/rstudio/shinyvalidate/issues

Encoding UTF-8

Imports shiny (>= 1.6), htmltools (>= 0.5.1.1), rlang (>= 0.4.10), glue (>= 1.4.2)

RoxygenNote 7.2.3

Suggests testthat, knitr, rmarkdown, covr

Config/testthat/edition 3

NeedsCompilation no

Author Carson Sievert [aut, cre] (<https://orcid.org/0000-0002-4958-2844>), Richard Iannone [aut] (<https://orcid.org/0000-0003-3925-190X>), Joe Cheng [aut], Posit Software, PBC [cph, fnd]

Maintainer Carson Sievert <carson@posit.co>

Repository CRAN

Date/Publication 2023-10-04 15:00:02 UTC

R topics documented:

`compose_rules` *Combine shinyvalidate rule functions*

Description

Takes multiple shinyvalidate rule functions, and returns a shinyvalidate rule function. When this resulting rule function is invoked, it will try each of its constituent rule functions in order; the first validation error that is detected will be returned immediately and the remaining rules will not be tried.

This function is not intended to be used by Shiny app authors (i.e. not for `InputValidator$add_rule("x", compose_rules(...))`), but for developers of reusable shinyvalidate rule functions. See examples.

Usage

```
compose_rules(...)
```

Arguments

`...`	Any number of shinyvalidate rule functions; earlier rules will be attempted before later rules. Argument names are ignored. Single-sided formulas are also accepted instead of a function, using `.` as the variable name for the input value.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

Other rule functions: `sv_between()`, `sv_email()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lte()`, `sv_lt()`, `sv_not_equal()`, `sv_numeric()`, `sv_optional()`, `sv_regex()`, `sv_required()`, `sv_url()`

Examples

```
# Create a new shinyvalidate rule that is composed
# of two `sv_*()` rule functions (`sv_integer()` and
# `sv_gt()`, and a custom function for ensuring
# a number is even)
positive_even_integer <- function() {
  compose_rules(
    sv_integer(),
    sv_gt(0),
    ~ if (. %% 2 == 1) "Must be an even number"
  )
}

# Use the `positive_even_integer()` rule function
# to check that a supplied value is an integer, greater
# than zero, and even (in that order)

## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("value", "Value")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Add two `add_rule()` statements: one that
  # combines `sv_required()` and `sv_numeric()` in
  # single rule, and another that is defined
  # through the use of `compose_rules()`
  iv$add_rule("value", compose_rules(sv_required(), sv_numeric()))
  iv$add_rule("value", positive_even_integer())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`InputValidator` *Shiny validation object*

Description

An R6 class for adding realtime input validation to Shiny apps.

`InputValidator` objects are designed to be created as local variables in Shiny server functions and Shiny module server functions. The Shiny app author can register zero, one, or multiple validation rules for each input field in their UI, using the `InputValidator$add_rule()` method.

Once an `InputValidator` object is created and populated with rules, it can be used in a few ways:

1. The `InputValidator$enable()` method can be called to display real-time feedback to users about what inputs are failing validation, and why.
2. The `InputValidator$is_valid()` method returns `TRUE` if and only if all of the validation rules are passing; this can be checked before executing actions that depend on the inputs being valid.
3. The `InputValidator$validate()` method is a lower-level feature that directly returns information about what fields failed validation, and why.

It's possible to have multiple `InputValidator` objects for each Shiny app. One scenario where this makes sense is if an app contains multiple forms that are completely unrelated to each other; each form would have its own `InputValidator` instance with a distinct set of rules.

Methods

Public methods:

Method `new()`**:** Create a new validator object.

Usage:

```
InputValidator$new(
  priority = 1000,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`priority` When a validator object is enabled, it creates an internal `shiny::observe()` to keep validation feedback in the UI up-to-date. This parameter controls the priority of that observer. It's highly recommended to keep this value higher than the priorities of any observers that do actual work, so users see validation updates quickly.

`session` The Shiny `session` object. (You should probably just use the default.)

Method `parent()`**:** For internal use only.

Usage:

```
InputValidator$parent(validator)
```

Arguments:

`validator` An `InputValidator` object.

Method `condition()`**:** Gets or sets a condition that overrides all of the rules in this validator. Before performing validation, this validator will execute the `cond` function. If `cond` returns `TRUE`, then validation continues as normal; if `FALSE`, then the validation rules will be skipped and treated as if they are all passing.

Usage:

```
InputValidator$condition(cond)
```

Arguments:

`cond` If this argument is missing, then the method returns the currently set condition function. If not missing, then `cond` must be either a zero-argument function that returns `TRUE` or `FALSE`; a single-sided formula that results in `TRUE` or `FALSE`; or `NULL` (which is equivalent to `~ TRUE`).

Returns: If `cond` is missing, then either `NULL` or a zero-argument function; if `cond` is provided, then nothing of consequence is returned.

Method `add_validator()`**:** Add another `InputValidator` object to this one, as a "child". Any time this validator object is asked for its validity, it will only return `TRUE` if all of its child validators are also valid; and when this validator object is enabled (or disabled), then all of its child validators are enabled (or disabled) as well.

This is intended to help with validating Shiny modules. Each module can create its own `InputValidator` object and populate it with rules, then return that object to the caller.

Usage:

```
InputValidator$add_validator(validator, label = deparse(substitute(validator)))
```

Arguments:

`validator` An `InputValidator` object.

`label` An optional label for the `InputValidator` object. By default, a label will be automatically generated.

Method `add_rule()`**:** Add an input validation rule. Each input validation rule applies to a single input. You can add multiple validation rules for a single input by calling `add_rule()` multiple times; the first validation rule for an input that fails will be used, and will prevent subsequent rules for that input from executing.

Usage:

```
InputValidator$add_rule(
  inputId,
  rule,
  ...,
  session. = shiny::getDefaultReactiveDomain()
)
```

Arguments:

`inputId` A single-element character vector indicating the ID of the input that this rule applies to. (Note that this name should *not* be qualified by a module namespace; e.g. pass `"x"` and not `session$ns("x")`.)

`rule` A function that takes (at least) one argument: the input's value. The function should return `NULL` if it passes validation, and if not, a single-element character vector or HTML tag containing an error message to display to the user near the input. You can alternatively provide a single-sided formula instead of a function, using `.` as the variable name for the input value being validated.

`...` Optional: Additional arguments to pass to the `rule` function whenever it is invoked.

`session`. The session object to which the input belongs. (There's almost never a reason to change this from the default.)

Method `enable()`**:** Begin displaying input validation feedback in the user interface. Once enabled, this validator object will automatically keep the feedback up-to-date. (It's safe to call the `enable()` method on an already-enabled validator.) If this validator object has been added to another validator object using `InputValidator$add_validator`, calls to `enable()` on this validator will be ignored.

Usage:

```
InputValidator$enable()
```

Method `disable()`**:** Clear existing input validation feedback in the user interface for all inputs represented in this validator's ruleset, and stop providing feedback going forward. Once disabled, `enable()` can be called to resume input validation.

Usage:

```
InputValidator$disable()
```

Method `fields()`**:** Returns `TRUE` if all input validation rules currently pass, `FALSE` if not.

Usage:

```
InputValidator$fields()
```

Method `is_valid()`**:** Returns `TRUE` if all input validation rules currently pass, `FALSE` if not.

Usage:

```
InputValidator$is_valid()
```

Method `validate()`**:** Run validation rules and gather results. For advanced usage only; most apps should use the `is_valid()` and `enable()` methods instead. The return value of this method is a named list, where the names are (fully namespace qualified) input IDs, and the values are either `NULL` (if the input value is passing) or a single-element character vector describing a validation problem.

Usage:

```
InputValidator$validate()
```

Method `_validate_impl()`**:** For internal use only.

Usage:

```
InputValidator$_validate_impl(indent)
```

Arguments:

`indent` For internal use only.

`input_provided` *Check whether an input value has been provided*

Description

This function takes an input value and uses heuristics to guess whether it represents an "empty" input vs. one that the user has provided. This will vary by input type; for example, a `shiny::textInput()` is `""` when empty, while a `shiny::numericInput()` is `NA`.

`input_provided` returns `TRUE` for all values except:

- `NULL`
- `""`
- An empty atomic vector or list
- An atomic vector that contains only missing (`NA`) values
- A character vector that contains only missing and/or `""` values
- An object of class `"try-error"`
- A value that represents an unclicked `shiny::actionButton()`

Usage

```
input_provided(val)
```

Arguments

`val`	Values to test for availability in a Shiny context.

Details

This function is based on `shiny::isTruthy()` but tweaked here in shinyvalidate to change the treatment of `FALSE` values: `isTruthy(FALSE)` returns `FALSE`, but `input_provided(FALSE)` returns `TRUE`. This difference is motivated by `shiny::checkboxInput()`, where `isTruthy()` answers the question of "is the input present *and checked*" while `input_provided` is just "is the input present".

Value

A logical vector of length 1.

skip_validation *Skip any normal validation performed by a rule*

Description

While the predominant role of the `skip_validation()` function is tied to the `sv_optional()` function (where it's used internally), you can also return `skip_validation()` from custom validation rules. When returned, all subsequent validation rules defined for the input will be skipped.

Usage

```
skip_validation()
```

Value

A function that returns a sentinel value, signaling to shinyvalidate that any further validation rules for an input are to be skipped.

sv_between *Validate that a field is a number bounded by minimum and maximum values*

Description

The `sv_between()` function validates that a field has values between left and right boundary values. Both bounds are inclusive by default, but both can be set as either inclusive or exclusive with the `inclusive` argument. In its default mode, the validation check will effectively be of the form `<left> <= <field> <= <right>`.

Usage

```
sv_between(
  left,
  right,
  inclusive = c(TRUE, TRUE),
  message_fmt = "Must be between {left} and {right}.",
  allow_na = FALSE,
  allow_nan = FALSE
)
```

Arguments

`left, right`	The left and right boundary values. Inclusively for each of the boundaries is set with the `inclusive` argument; the defaults are set for inclusive bounds.
`inclusive`	A two-element logical vector that indicates whether the `left` and `right` bounds, respectively, should be inclusive. Both bounds are by default are inclusive, using `c(TRUE, TRUE)`.
`message_fmt`	The validation error message to use if a value fails to match the rule. The message can be customized by using the `"{left}"` and `"{right}"` string parameters, which allows for the insertion of the `left` and `right` values. While the default message uses both of these string parameters, they are not required in a user-defined `message_fmt` string.
`allow_na, allow_nan`	If `FALSE` (the default for both options), then any `NA` or `NaN` element will cause validation to fail.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The `sv_in_set()` function, which tests whether a field values are part of a specified set.

Other rule functions: `compose_rules()`, `sv_email()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lte()`, `sv_lt()`, `sv_not_equal()`, `sv_numeric()`, `sv_optional()`, `sv_regex()`, `sv_required()`, `sv_url()`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("count", "Count")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_between()` requires `left` and
  # `right` boundary values; a message will be
  # displayed if the validation of `input$count` fails
  iv$add_rule("count", sv_between(10, 100))
```

```
  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_email` *Validate that a field contains an email address*

Description

A validation function, suitable for use with `InputValidator$add_rule()`, that checks whether an input value looks like a valid email address.

Usage

```
sv_email(
  message = "Not a valid email address",
  allow_multiple = FALSE,
  allow_na = FALSE
)
```

Arguments

`message`	The validation error message to use if a value doesn't match a regex pattern for email address detection.
`allow_multiple`	If `FALSE` (the default), then the length of the input vector must be exactly one; if `TRUE`, then any length is allowed (including a length of zero; use `sv_required()` if one or more values should be required).
`allow_na`	If `FALSE`, then any `NA` element will cause validation to fail.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The `sv_url()` function, another specialized regex-based function for validating URLs. For general regex-based validation the `sv_regex()` function is useful.

Other rule functions: `compose_rules()`, `sv_between()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lte()`, `sv_lt()`, `sv_not_equal()`, `sv_numeric()`, `sv_optional()`, `sv_regex()`, `sv_required()`, `sv_url()`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("email", "Email")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_email()` works well with its
  # defaults; a message will be displayed if the
  # validation of `input$email` fails
  iv$add_rule("email", sv_email())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_equal` *Validate that a field is equal to a specified value*

Description

The `sv_equal()` function compares the field value to a specified value with the == operator.

Usage

```
sv_equal(
  rhs,
  message_fmt = "Must be equal to {rhs}.",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

rhs	The right hand side (RHS) value is to be used for the comparison with the field value. The validation check will effectively be of the form <field> == <rhs>.
message_fmt	The validation error message to use if the field fails the validation test. Use the "{rhs}" string parameter to customize the message, including what was set in rhs. While the default message uses this string parameter, it is not required in a user-defined message_fmt string.
allow_multiple	If FALSE (the default), then the length of the input vector must be exactly one; if TRUE, then any length is allowed (including a length of zero; use sv_required() if one or more values should be required).
allow_na, allow_nan	If FALSE (the default for both options), then any NA or NaN element will cause validation to fail.
allow_inf	If FALSE (the default), then any Inf or -Inf element will cause validation to fail.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The other comparison-based rule functions: sv_gt(), sv_gte(), sv_lt(), sv_lte(), and sv_not_equal() (which serves as the opposite function to sv_equal()).

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_gte(), sv_gt(), sv_in_set(), sv_integer(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("number", "Number")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_equal()` requires a value
  # to compare against the field value; a message
  # will be shown if the validation of
  # `input$number` fails
```

```
  iv$add_rule("number", sv_equal(1))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

---

sv_gt *Validate that a field is greater than a specified value*

Description

The `sv_gt()` function compares the field value to a specified value with the > operator.

Usage

```
sv_gt(
  rhs,
  message_fmt = "Must be greater than {rhs}.",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

`rhs`	The right hand side (RHS) value is to be used for the comparison with the field value. The validation check will effectively be of the form `<field> > <rhs>`.
`message_fmt`	The validation error message to use if the field fails the validation test. Use the `"{rhs}"` string parameter to customize the message, including what was set in `rhs`. While the default message uses this string parameter, it is not required in a user-defined `message_fmt` string.
`allow_multiple`	If `FALSE` (the default), then the length of the input vector must be exactly one; if `TRUE`, then any length is allowed (including a length of zero; use `sv_required()` if one or more values should be required).
`allow_na, allow_nan`	If `FALSE` (the default for both options), then any `NA` or `NaN` element will cause validation to fail.
`allow_inf`	If `FALSE` (the default), then any `Inf` or `-Inf` element will cause validation to fail.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The other comparison-based rule functions: sv_gte(), sv_lt(), sv_lte(), sv_equal(), and sv_not_equal(). The sv_gte() function may be needed if the field value should also pass validation when equal to the comparison value.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_in_set(), sv_integer(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("number", "Number")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_gt()` requires a value
  # to compare against the field value; a message
  # will be shown if the validation of
  # `input$number` fails
  iv$add_rule("number", sv_gt(0))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

---

sv_gte *Validate that a field is greater than or equal to a specified value*

Description

The sv_gte() function compares the field value to a specified value with the >= operator.

Usage

```
sv_gte(
  rhs,
  message_fmt = "Must be greater than or equal to {rhs}.",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

rhs	The right hand side (RHS) value is to be used for the comparison with the field value. The validation check will effectively be of the form <field> >= <rhs>.
message_fmt	The validation error message to use if the field fails the validation test. Use the "{rhs}" string parameter to customize the message, including what was set in rhs. While the default message uses this string parameter, it is not required in a user-defined message_fmt string.
allow_multiple	If FALSE (the default), then the length of the input vector must be exactly one; if TRUE, then any length is allowed (including a length of zero; use sv_required() if one or more values should be required).
allow_na, allow_nan	If FALSE (the default for both options), then any NA or NaN element will cause validation to fail.
allow_inf	If FALSE (the default), then any Inf or -Inf element will cause validation to fail.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The other comparison-based rule functions: sv_gt(), sv_lt(), sv_lte(), sv_equal(), and sv_not_equal(). The sv_gt() function may be needed if the field value should not pass validation when it is equal to the comparison value.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gt(), sv_in_set(), sv_integer(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
```

```
  textInput("number", "Number")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_gte()` requires a value
  # to compare against the field value; a message
  # will be shown if the validation of
  # `input$number` fails
  iv$add_rule("number", sv_gte(1))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_integer` *Validate that a field is a number that is integer-like*

Description

The `sv_integer()` function validates that a field is 'integer-like' with the `{value} %% 1 == 0` test. Very large values (generally with absolute exponent values greater than 15) won't be validated correctly due to floating point imprecision. By default, only a single, finite, not-missing, valid numbers are allowed, but each of those criteria can be controlled via arguments.

Usage

```
sv_integer(
  message = "An integer is required",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE
)
```

Arguments

`message`	The validation error message to use if a value is not an integer.
`allow_multiple`	If `FALSE` (the default), then the length of the input vector must be exactly one; if `TRUE`, then any length is allowed (including a length of zero; use `sv_required()` if one or more values should be required).

allow_na, allow_nan
: If FALSE (the default for both options), then any NA or NaN element will cause validation to fail.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The sv_numeric() function, which tests whether a field value is simply numeric.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_gt(), sv_in_set(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("count", "Count")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_integer()` works well with its
  # defaults; a message will be displayed if the
  # validation of `input$count` fails
  iv$add_rule("count", sv_integer())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

sv_in_set *Validate that a field is part of a defined set*

Description

The sv_in_set() function checks whether the field value is a member of a specified set of values.

Usage

```
sv_in_set(
  set,
  message_fmt = "Must be in the set of {values_text}.",
  set_limit = 3
)
```

Arguments

set	A vector or list of elements for which the field value must be a part of (value %in% set must be TRUE) to pass validation. To allow an empty field, NA should be included in the set vector. Optionally, NaN can be included as well.
message_fmt	The validation error message to use if a value fails to match the rule. The message can be customized by using the "{values_text}" string parameter, which allows for the insertion of set values (formatted internally as a text list and controlled via the set_limit parameter). While the default message uses this string parameter, it is not required in a user-defined message_fmt string.
set_limit	The limit of set values to include in the automatically-generated error message (i.e., when message = NULL, the default). If the number of elements provided in set is greater than set_limit then only the first <message_limit> set elements will be echoed along with text that states how many extra elements are part of the set.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The sv_between() function, which tests whether a field values between two boundary values.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_gt(), sv_integer(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("rating", "Rating")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_in_set()` requires a value
  # set given as a vector; a message will be
  # shown if the validation of `input$rating` fails
  iv$add_rule("rating", sv_in_set(1:5))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

---

sv_lt *Validate that a field is less than a specified value*

Description

The `sv_lt()` function compares the field value to a specified value with the < operator.

Usage

```
sv_lt(
  rhs,
  message_fmt = "Must be less than {rhs}.",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

rhs	The right hand side (RHS) value is to be used for the comparison with the field value. The validation check will effectively be of the form <field> < <rhs>.
message_fmt	The validation error message to use if the field fails the validation test. Use the "{rhs}" string parameter to customize the message, including what was set in rhs. While the default message uses this string parameter, it is not required in a user-defined message_fmt string.
allow_multiple	If FALSE (the default), then the length of the input vector must be exactly one; if TRUE, then any length is allowed (including a length of zero; use sv_required() if one or more values should be required).
allow_na, allow_nan	If FALSE (the default for both options), then any NA or NaN element will cause validation to fail.
allow_inf	If FALSE (the default), then any Inf or -Inf element will cause validation to fail.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The other comparison-based rule functions: sv_gt(), sv_gte(), sv_lte(), sv_equal(), and sv_not_equal(). The sv_lte() function may be needed if the field value should also pass validation when equal to the comparison value.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_gt(), sv_in_set(), sv_integer(), sv_lte(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("number", "Number")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_lt()` requires a value
  # to compare against the field value; a message
  # will be shown if the validation of
```

```
  # `input$number` fails
  iv$add_rule("number", sv_lt(10))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_lte` *Validate that a field is less than or equal to a specified value*

Description

The `sv_lte()` function compares the field value to a specified value with the `<=` operator.

Usage

```
sv_lte(
  rhs,
  message_fmt = "Must be less than or equal to {rhs}.",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

`rhs`	The right hand side (RHS) value is to be used for the comparison with the field value. The validation check will effectively be of the form `<field> <= <rhs>`.
`message_fmt`	The validation error message to use if the field fails the validation test. Use the `"{rhs}"` string parameter to customize the message, including what was set in `rhs`. While the default message uses this string parameter, it is not required in a user-defined `message_fmt` string.
`allow_multiple`	If `FALSE` (the default), then the length of the input vector must be exactly one; if `TRUE`, then any length is allowed (including a length of zero; use `sv_required()` if one or more values should be required).
`allow_na, allow_nan`	If `FALSE` (the default for both options), then any `NA` or `NaN` element will cause validation to fail.
`allow_inf`	If `FALSE` (the default), then any `Inf` or `-Inf` element will cause validation to fail.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The other comparison-based rule functions: `sv_gt()`, `sv_gte()`, `sv_lt()`, `sv_equal()`, and `sv_not_equal()`. The `sv_lt()` function may be needed if the field value should not pass validation when it is equal to the comparison value.

Other rule functions: `compose_rules()`, `sv_between()`, `sv_email()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lt()`, `sv_not_equal()`, `sv_numeric()`, `sv_optional()`, `sv_regex()`, `sv_required()`, `sv_url()`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("number", "Number")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_lte()` requires a value
  # to compare against the field value; a message
  # will be shown if the validation of
  # `input$number` fails
  iv$add_rule("number", sv_lte(0))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_not_equal` *Validate that a field is not equal to a specified value*

Description

The sv_not_equal() function compares the field value to a specified value with the != operator.

Usage

```
sv_not_equal(
  rhs,
  message_fmt = "Must not be equal to {rhs}.",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

rhs	The right hand side (RHS) value is to be used for the comparison with the field value. The validation check will effectively be of the form <field> != <rhs>.
message_fmt	The validation error message to use if the field fails the validation test. Use the "{rhs}" string parameter to customize the message, including what was set in rhs. While the default message uses this string parameter, it is not required in a user-defined message_fmt string.
allow_multiple	If FALSE (the default), then the length of the input vector must be exactly one; if TRUE, then any length is allowed (including a length of zero; use sv_required() if one or more values should be required).
allow_na, allow_nan	If FALSE (the default for both options), then any NA or NaN element will cause validation to fail.
allow_inf	If FALSE (the default), then any Inf or -Inf element will cause validation to fail.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The other comparison-based rule functions: sv_gt(), sv_gte(), sv_lt(), sv_lte(), and sv_equal() (which serves as the opposite function to sv_not_equal()).

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_gt(), sv_in_set(), sv_integer(), sv_lte(), sv_lt(), sv_numeric(), sv_optional(), sv_regex(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
```

```
library(shinyvalidate)

ui <- fluidPage(
  textInput("score", "Number")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_not_equal()` requires a value
  # to compare against the field value; a message
  # will be shown if the validation of
  # `input$score` fails
  iv$add_rule("score", sv_not_equal(0))

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_numeric` *Validate that a field is a number*

Description

The `sv_numeric()` function validates that a field is numeric with the `base::is.numeric()` function. By default, only a single, finite, not-missing, valid number is allowed, but each of those criteria can be controlled via arguments.

Usage

```
sv_numeric(
  message = "A number is required",
  allow_multiple = FALSE,
  allow_na = FALSE,
  allow_nan = FALSE,
  allow_inf = FALSE
)
```

Arguments

`message`	The validation error message to use if a value is not numeric.

`allow_multiple` If `FALSE` (the default), then the length of the input vector must be exactly one; if `TRUE`, then any length is allowed (including a length of zero; use `sv_required()` if one or more values should be required).

`allow_na, allow_nan` If `FALSE` (the default for both options), then any `NA` or `NaN` element will cause validation to fail.

`allow_inf` If `FALSE` (the default), then any `Inf` or `-Inf` element will cause validation to fail.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The `sv_integer()` function, which tests whether a field value is a number that is integer-like.

Other rule functions: `compose_rules()`, `sv_between()`, `sv_email()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lte()`, `sv_lt()`, `sv_not_equal()`, `sv_optional()`, `sv_regex()`, `sv_required()`, `sv_url()`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("rating", "Rating")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_numeric()` works well with its
  # defaults; a message will be displayed if the
  # validation of `input$rating` fails
  iv$add_rule("rating", sv_numeric())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_optional` *Indicate that a field is optional*

Description

Call `sv_optional()` to generate a validation function that indicates an input is allowed to *not* be present. If an `sv_optional()` rule sees that an input is not present, subsequent rules for that input are skipped and the input is considered valid. Otherwise, the rule simply passes. (`sv_optional()` will never return a validation error/message.)

By default, the definition of "is present" is based on `input_provided()`.

Child validators (see `InputValidator$add_validator()`) are not affected by `sv_optional()` rules in parent validators; only rules in the same validator instance as the `sv_optional()` will be skipped.

Usage

```
sv_optional(test = input_provided)
```

Arguments

`test`	A single-argument function, or single-sided formula (using `.` to access the value to test), that returns `TRUE` for success and `FALSE` for failure.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The `sv_required()` function, which takes a different approach to field presence.

Other rule functions: `compose_rules()`, `sv_between()`, `sv_email()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lte()`, `sv_lt()`, `sv_not_equal()`, `sv_numeric()`, `sv_regex()`, `sv_required()`, `sv_url()`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("email", "Email")
)

server <- function(input, output, session) {
```

```
  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_optional()` is often paired with
  # another `sv_*()` function; below, an email in
  # `input$email` is not required, but if present, it
  # must be valid
  iv$add_rule("email", sv_optional())
  iv$add_rule("email", sv_email())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

`sv_regex` *Validate that a field matches a regular expression*

Description

A validation function, suitable for use with `InputValidator$add_rule()`, that checks whether input values match the specified regular expression.

Usage

```
sv_regex(
  pattern,
  message,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE,
  invert = FALSE
)
```

Arguments

`pattern`	Character string containing a regular expression (or character string if `fixed = TRUE`) to be tested against. If a character vector of length 2 or more is supplied, the first element is used with a warning.
`message`	The validation error message to use if a value fails to match the pattern.
`ignore.case, perl, fixed, useBytes, invert`	Options passed through to `base::grepl()`.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The sv_email() and sv_url() functions, which are specialized regex-based functions for validating email addresses and URLs.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_gt(), sv_in_set(), sv_integer(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_required(), sv_url()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("lookup_id", "Lookup ID")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_regex()` requires both a regex
  # pattern and message to display if the validation
  # of `input$lookup_id` fails
  iv$add_rule(
    "lookup_id",
    sv_regex("^[a-zA-Z0-9]*$", "Only alphanumeric characters allowed")
  )

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}

# As an alternative to the above example, the
# following snippet can serve to replace the
# `iv$add_rule(...)` statement

# If you're more comfortable with wildcards
# (i.e., globbing) than with regular expressions,
```

```
# use `glob2rx()` in `pattern`

# iv$add_rule(
#   "lookup_id",
#   sv_regex(
#     pattern = glob2rx("*.png"),
#     message = "A filename ending in 'png' was expected",
#     ignore.case = TRUE
#   )
# )
```

`sv_required` *Validate that the field is present*

Description

Call `sv_required()` to generate a validation function that ensures an input value is present. By default, the definition of "is present" is based on `input_provided()`.

Usage

```
sv_required(message = "Required", test = input_provided)
```

Arguments

`message`	The validation error message to be displayed if the test does not pass.
`test`	A single-argument function, or single-sided formula (using `.` to access the value to test), that returns `TRUE` for success and `FALSE` for failure.

Value

A function suitable for use as an `InputValidator$add_rule()` rule.

See Also

The `sv_optional()` function, which takes a different approach to field presence.

Other rule functions: `compose_rules()`, `sv_between()`, `sv_email()`, `sv_equal()`, `sv_gte()`, `sv_gt()`, `sv_in_set()`, `sv_integer()`, `sv_lte()`, `sv_lt()`, `sv_not_equal()`, `sv_numeric()`, `sv_optional()`, `sv_regex()`, `sv_url()`

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)
```

```
ui <- fluidPage(
  textInput("name", "Name")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: ensure that `input$name` is present,
  # and return a terse validation message if not
  iv$add_rule("name", sv_required())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}

# There are some alternatives to the above example,
# and the following snippets can serve to replace
# the `iv$add_rule(...)` statement

# (1) Providing a custom message to display
# when validation fails:

# iv$add_rule("email", sv_required("An email is required"))

# (2) Providing a `test` argument to change
# the definition of "is present"; in this
# snippet, any non-NULL value will be accepted:

# iv$add_rule("choices", sv_required(test = is.null))
```

`sv_url` *Validate that a field contains a URL*

Description

A validation function, suitable for use with `InputValidator$add_rule()`, that checks whether an input value is a valid URL.

Usage

```
sv_url(message = "Not a valid URL", allow_multiple = FALSE, allow_na = FALSE)
```

Arguments

message	The validation error message to use if a value doesn't match a regex pattern for URL detection.
allow_multiple	If FALSE (the default), then the length of the input vector must be exactly one; if TRUE, then any length is allowed (including a length of zero; use sv_required() if one or more values should be required).
allow_na	If FALSE, then any NA element will cause validation to fail.

Value

A function suitable for use as an InputValidator$add_rule() rule.

See Also

The sv_email() function, another specialized regex-based function for validating email addresses. For general regex-based validation the sv_regex() function is useful.

Other rule functions: compose_rules(), sv_between(), sv_email(), sv_equal(), sv_gte(), sv_gt(), sv_in_set(), sv_integer(), sv_lte(), sv_lt(), sv_not_equal(), sv_numeric(), sv_optional(), sv_regex(), sv_required()

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {

library(shiny)
library(shinyvalidate)

ui <- fluidPage(
  textInput("url", "URL")
)

server <- function(input, output, session) {

  # Validation rules are set in the server, start by
  # making a new instance of an `InputValidator()`
  iv <- InputValidator$new()

  # Basic usage: `sv_url()` works well with its
  # defaults; a message will be displayed if the
  # validation of `input$address` fails
  iv$add_rule("url", sv_url())

  # Finally, `enable()` the validation rules
  iv$enable()
}

shinyApp(ui, server)

}
```

Index