

# Package ‘tutorial.helpers’

June 26, 2024

**Title** Helper Functions for Creating Tutorials

**Version** 0.3.0

**Description** Helper functions for creating, editing, and testing tutorials created with the 'learnr' package. Provides a simple method for allowing students to download their answers to tutorial questions. For examples of its use, see the 'r4ds.tutorials' and 'primer.tutorials' packages.

**Depends** R (>= 4.1.0)

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**VignetteBuilder** knitr

**Suggests** knitr, roxygen2, rsconnect, shinytest, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Imports** dplyr, learnr, mime, parsermd, purrr, readr, rmarkdown, rstudioapi, rvest, shiny, stringr, tibble

**BugReports** <https://github.com/PPBDS/tutorial.helpers/issues>

**URL** <https://ppbds.github.io/tutorial.helpers/>,  
<https://github.com/PPBDS/tutorial.helpers>

**NeedsCompilation** no

**Author** David Kane [aut, cre, cph] (<<https://orcid.org/0000-0002-6660-3934>>)

**Maintainer** David Kane <dave.kane@gmail.com>

**Repository** CRAN

**Date/Publication** 2024-06-26 19:00:02 UTC

## Contents

check_current_tutorial . . . . .	2
check_tutorial_defaults . . . . .	2
determine_code_chunk_name . . . . .	3

determine_exercise_number . . . . .	3
format_tutorial . . . . .	4
get_submissions_from_learnr_session . . . . .	4
knit_tutorials . . . . .	5
make_exercise . . . . .	5
process_submissions . . . . .	6
return_tutorial_paths . . . . .	7
set_binary_only_in_r_profile . . . . .	8
set_rstudio_settings . . . . .	8
show_file . . . . .	9
submission_server . . . . .	10
write_answers . . . . .	11

<b>Index</b>	<b>12</b>
--------------	-----------

---

check\_current\_tutorial

*Check current tutorial*

---

### Description

An add-in for formatting tutorials.

Uses format\_tutorial() to format the tutorial Rmd open in the current editor

### Usage

check\_current\_tutorial()

---

check\_tutorial\_defaults

*Confirm that a tutorial has the recommended components*

---

### Description

There are three code components: the use of a copy-code button, an information request, and a download page. It is tricky to know where to store the "truth" of what these components should look like. For now, the truth is defined as the skeleton.Rmd which defines the template for creating a new tutorial.

All tutorials should also have library(learnr) and library(tutorial.helpers), both of which exist in the skeleton

### Usage

check\_tutorial\_defaults(tutorial\_paths)

**Arguments**

tutorial\_paths Character vector of the paths to the tutorials to be examined.

**Value**

No return value, called for side effects.

**Examples**

```
check_tutorial_defaults(tutorial_paths = return_tutorial_paths("tutorial.helpers"))
```

---

determine\_code\_chunk\_name

*Determine the code chunk name of a new exercise in a tutorial.*

---

**Description**

Determine the code chunk name of a new exercise in a tutorial.

**Usage**

```
determine_code_chunk_name(file_path = NULL)
```

**Arguments**

file\_path Character string of the file path to the tutorial

**Value**

The section id of the exercise based on its section

---

determine\_exercise\_number

*Finds the number of the next exercise in a tutorial*

---

**Description**

Finds the number of the next exercise in a tutorial

**Usage**

```
determine_exercise_number(file_path = NULL)
```

**Arguments**

file\_path Character string of the file path to the tutorial

**Value**

The next exercise number based on the file argument or the active document.

---

<code>format_tutorial</code>	<i>Re-format a tutorial</i>
------------------------------	-----------------------------

---

**Description**

A function for formatting tutorial Rmd files. Used by `check_current_tutorial()` to re-format the currently open tutorial in RStudio. It renumbers the exercises so that they are in order. It ensures that chunk labels use this numbering, along with the section title.

**Usage**

```
format_tutorial(file_path)
```

**Arguments**

`file_path`      Character string.

**Value**

Formatted document with correct exercise, hint and test chunk labels.

---

<code>get_submissions_from_learnr_session</code>	<i>Return a list of tutorial answers</i>
--	--

---

**Description**

Grabs information from the `learnr` session environment, not directly from the session object itself. Since we are using the session environment, we currently don't (?) have a way to save the environment and hence can't test this function.

**Usage**

```
get_submissions_from_learnr_session(sess)
```

**Arguments**

`sess`              session object from shiny with `learnr`

**Value**

a list which includes the exercise submissions of tutorial

---

knit_tutorials	<i>Knit a set of tutorials</i>
----------------	--------------------------------

---

**Description**

We define "testing" a tutorial as (successfully) running `render()` on it. This function renders all the tutorials provided in `tutorial_paths`. There is no check to see if the rendered file looks OK. If a tutorial fails to render, then (we assume!) an error will be generated which will then filter up to our testing rig.

**Usage**

```
knit_tutorials(tutorial_paths)
```

**Arguments**

`tutorial_paths` Character vector of the paths to the tutorials to be knitted.

**Value**

No return value, called for side effects.

**Examples**

```
knit_tutorials(tutorial_paths = return_tutorial_paths("tutorial.helpers"))
```

---

make_exercise	<i>Add a tutorial code exercise or question to the active document</i>
---------------	--

---

**Description**

When writing tutorials, it is handy to be able to insert the skeleton for a new code exercise or question. We bind `make_exercise()` and friends as an RStudio add-in to provide this functionality. Note that the function determines the correct exercise number to use and also adds appropriate code chunk names, based on the exercise number and section title.

**Usage**

```
make_exercise(type = "code", file_path = NULL)
```

```
make_no_answer()
```

```
make_yes_answer()
```

**Arguments**

type	Character of question type. Must be one of "code", "no-answer", or "yes-answer".
file_path	Character path to a file. If NULL, the RStudio active document is used, which is the default behavior. An actual file path is used for testing.

**Details**

It appears that the RStudio addins must have function names only as the Binding value. In other words, you can't have `make_exercise(type = 'no-answer')` as the value. So, we need two extra functions — `make_no_answer()` and `make_yes_answer()` — which just call `make_exercise()` while passing in the correct argument.

**Value**

Exercise skeleton corresponding to the type argument.

---

process\_submissions    *Process Submissions*

---

**Description**

This function processes submissions from a directory containing HTML/XML files. It extracts tables from the files, filters them based on a pattern and key variables, and returns either a summary tibble or a combined tibble with all the data.

**Usage**

```
process_submissions(
  path,
  pattern = ".",
  return_value = "Summary",
  key_vars = NULL,
  verbose = 0,
  keep_file_name = NULL
)
```

**Arguments**

path	The path to the directory containing the HTML/XML files.
pattern	The pattern to match against the file names (default: ".").
return_value	The type of value to return. Allowed values are "Summary" (default) or "All".
key_vars	A character vector of key variables to extract from the "id" column (default: NULL).

**verbose** An integer specifying the verbosity level. 0: no messages, 1: file count messages, 2: some detailed messages about files, 3: detailed messages including all file problems (default: 0).

**keep\_file\_name** Specifies whether to keep the file name in the summary tibble. Allowed values are NULL (default), "All" (keep entire file name), "Space" (keep up to first space), or "Underscore" (keep up to first underscore). Only used when return\_value is "Summary".

### Value

If return\_value is "Summary", returns a tibble with one row for each file, columns corresponding to the key\_vars, and an additional "answers" column indicating the number of rows in each tibble. If return\_value is "All", returns a tibble with all the data combined from all the files.

### Examples

```
## Not run:
# Process submissions with default settings
process_submissions("path/to/directory")

# Process submissions with a specific pattern and key variables
process_submissions("path/to/directory", pattern = "^submission", key_vars = c("name", "email"))

# Process submissions and return all data
process_submissions("path/to/directory", return_value = "All")

# Process submissions with verbose output (level 3)
process_submissions("path/to/directory", verbose = 3)

# Process submissions and keep the entire file name in the summary tibble
process_submissions("path/to/directory", return_value = "Summary", keep_file_name = "All")

## End(Not run)
```

---

return\_tutorial\_paths *Return all the paths to the tutorials in a package*

---

### Description

Takes a package name and returns a character vector of all the paths to tutorials in the installed package. Assumes that every Rmd file in inst/tutorials/\* is a tutorial, which should be true.

### Usage

```
return_tutorial_paths(package)
```

### Arguments

**package** Character vector of the package name to be tested.

**Value**

Character vector of the full paths to all installed tutorials in package.

**Examples**

```
return_tutorial_paths('learnr')
```

---

```
set_binary_only_in_r_profile
```

*Set pkgType to binary in .Rprofile*

---

**Description**

This function sets the `pkgType` global option to "binary" in your `.Rprofile`. New R users, especially those on Windows, should never install from source. Doing so fails too often, and too confusingly. It also sets the value for this R session. So, you do not need to either restart R nor source the `.Rprofile` by hand.

You can examine your `.Rprofile` to confirm this change with `usethis::edit_r_profile()`

**Usage**

```
set_binary_only_in_r_profile()
```

**Value**

No return value, called for side effects.

---

```
set_rstudio_settings Select smart setting for RStudio
```

---

**Description**

This function changes RStudio settings in order to make learning easier for new users. These settings are stored in: `~/config/rstudio/rstudio-prefs.json`. The most important changes are `save_workspace` to "never", `load_workspace` to FALSE, and `insert_native_pipe_operator` to TRUE. All those changes are good for any user, new or old.

We also change `rmd_viewer_type` to "pane", `show_hidden_files` to TRUE, `rmd_chunk_output_inline` to FALSE, `source_with_echo` to TRUE, and `packages_pane_enabled` to FALSE. These settings make RStudio less confusing to new users. The `rmd_viewer_type` setting is especially useful to students copy/pasting from the Console/Terminal to a tutorial.

The last two changes are setting both `rainbow_parentheses` and `syntax_color_console` to TRUE. We *think* that these settings make coding errors less likely.

**Usage**

```
set_rstudio_settings(set.binary = TRUE)
```

**Arguments**

`set.binary` Logical, set to TRUE, which indicates whether or not `set_binary_only_in_r_profile()` should be run at the end.

**Value**

No return value, called for side effects.

---

<code>show_file</code>	<i>Display the contents of a text file that match a pattern</i>
------------------------	---

---

**Description**

This function reads the contents of a text file and either prints the specified range of rows that match a given regular expression pattern or prints the code lines within R code chunks. If `start` is a negative number, it prints the last `abs(start)` lines, ignoring missing lines at the end of the file.

**Usage**

```
show_file(path, start = 1, end = NULL, pattern = NULL, chunk = "None")
```

**Arguments**

`path` A character vector representing the path to the text file.

`start` An integer specifying the starting row number (inclusive) to consider. Default is 1. If negative, it represents the number of lines to print from the end of the file.

`end` An integer specifying the ending row number (inclusive) to consider. Default is the last row.

`pattern` A regular expression pattern to match against each row. Default is NULL (no pattern matching).

`chunk` A character string indicating whether to print code lines within R code chunks. Possible values are "None" (default), "All" (print all code chunks), or "Last" (print only the last code chunk).

**Value**

The function prints the contents of the specified range of rows that match the pattern (if provided) or the code lines within R code chunks (if `chunk` is TRUE) to the console. If no rows match the pattern, nothing is printed. If `start` is negative, the function prints the last `abs(start)` lines, ignoring missing lines at the end of the file.

## Examples

```
## Not run:
# Display all rows of a text file
show_file("path/to/your/file.txt")

# Display rows 5 to 10 of a text file
show_file("path/to/your/file.txt", start = 5, end = 10)

# Display all rows of a text file that contain the word "example"
show_file("path/to/your/file.txt", pattern = "example")

# Print code lines within R code chunks
show_file("path/to/your/file.txt", chunk = TRUE)

# Display the last 5 lines of a text file, ignoring missing lines at the end
show_file("path/to/your/file.txt", start = -5)

## End(Not run)
```

---

submission\_server

*Tutorial submission functions*

---

## Description

The following function was modified from Colin Rundel's `learnrhash` package, available at <https://github.com/rundel/learnrhash>. Note that when including these functions in a learnr Rmd document it is necessary that the server function, `submission_server()`, be included in an R chunk where `context="server"`.

## Usage

```
submission_server()
```

```
submission_ui
```

## Format

An object of class `shiny.tag` of length 3.

## Value

No return value, called for side effects.

An object of class `shiny.tag`.

**Examples**

```
if(interactive()){
  submission_server()
}

if(interactive()){
  submission_ui
}
```

---

write_answers	<i>Write tutorial answers to file</i>
---------------	---------------------------------------

---

**Description**

Take a tutorial session, extract out all the submitted answers, and write out an html file with all of those answers.

**Usage**

```
write_answers(file, session, is_test = FALSE)
```

**Arguments**

file	Location to render answers to. Output file type determined by file suffix. Only "html" is acceptable.
session	Session object from Shiny with learnr.
is_test	TRUE/FALSE depending on whether or not we are just testing the function. Default is TRUE.

**Details**

We only keep track of the questions/exercises that the student has completed. The other obvious approach is to keep all the questions/exercises and leave unanswered ones as NA. Not sure if that approach is better, or even possible.

**Examples**

```
if(interactive()){
  write_answers("getting-started_answers.html", sess)
}
```

# Index

## \* datasets

- submission\_server, 10
  
- check\_current\_tutorial, 2
- check\_tutorial\_defaults, 2
  
- determine\_code\_chunk\_name, 3
- determine\_exercise\_number, 3
  
- format\_tutorial, 4
  
- get\_submissions\_from\_learnr\_session, 4
  
- knit\_tutorials, 5
  
- make\_exercise, 5
- make\_no\_answer (make\_exercise), 5
- make\_yes\_answer (make\_exercise), 5
  
- process\_submissions, 6
  
- return\_tutorial\_paths, 7
  
- set\_binary\_only\_in\_r\_profile, 8
- set\_rstudio\_settings, 8
- show\_file, 9
- submission\_server, 10
- submission\_ui (submission\_server), 10
  
- write\_answers, 11