Package ‘TexExamRandomizer’

February 13, 2018

Type Package
Title Personalizes and Randomizes Exams Written in 'LaTeX'
Version 1.2.3
Author Alejandro Gonzalez Recuenco
Maintainer Alejandro Gonzalez Recuenco <alejandrogonzalezrecuenco@gmail.com>
Description Randomizing exams with 'LaTeX'.
   If you can compile your main document with 'LaTeX', the program should be able to com-
   pile the randomized
   versions without much extra effort when creating the document.
URL https://github.com/alexrecuenco/TexExamRandomizer
BugReports https://github.com/alexrecuenco/TexExamRandomizer/issues
Encoding UTF-8
LazyData true
Imports Rcpp (>= 0.12.13), assertthat, stringr, jsonlite, stats, utils
Depends
Suggests optparse, knitr, rmarkdown
License MIT + file LICENSE
LinkingTo Rcpp
RoxygenNote 6.0.1
ByteCompile true
VignetteBuilder knitr
SystemRequirements C++11, A modern compiler (>=gcc-4.9), And latexmk
   is necessary to compile all output documents with the functions
   provided by this package
NeedsCompilation yes
Repository CRAN
Date/Publication 2018-02-13 04:13:42 UTC
Description

This package is designed with exams and homework created in 'LaTeX' in mind. It allows to randomize and personalize exams and homework and it aids the user with grading them.

Details

If you are using the exam class from 'LaTeX' already, it is likely that this program works as it is.

If you just want to randomize your exams,

- Look at vignette("BasicUse", package = "TexExamRandomizer") for an introduction of the concept behind this library and a quick way to start using it.

- Look at vignette("ExamOptions", package = "TexExamRandomizer") for a more detailed explanations of what options can be used on a document.

If instead you are trying to use the library to create your own randomizer for a certain use you might have, you should start by looking at `CreateRandomExams` and `GenerateHomework`. 
Description
Behaves like `cat`, but it first automatically unlists the exam to print the document. Since the document is kept as a tree of lists, it simply abstract the idea of outputting the document with one document.

Usage
```
catDocument(FullDocument, sep = "\n", ...)  
```

Arguments
- `FullDocument` Document as structure by `StructureDocument`
- `sep` The separation character(s) between each line.
- `...` all extra arguments get passed along to the command "cat"

Examples
```
catDocument(TexExamRandomizer::testdoc)  
```
Description

Constructs an answer sheet given a document as generated by `StructureDocument` by finding in the items the correct and wrong tags and describing where it found them.

Note that you must provide the document part only, `StructureDocument` gives back a `$preamble` and `$document`.

If `wrongTag` is left `NULL`, the answer sheet only shows information of the correct answers.

This answer sheet provides information for what answers are correct or incorrect, as well as their position within the original document, before any shuffling was done. (It uses the names of the document to decide whether the document was shuffled or not, since subsetting a list removes all attributes except for the names, this is the "safest" way to do it)

The intent of this function is to make it easy to find the answers for a randomized version of an exam.

Usage

`ConstructAnswerSheet(document, correctTag, wrongTag = NULL)`

Arguments

- `Document`: Document, as defined in `StructureDocument`. Remember however that the function `StructureDocument` returns the document and the preamble together in a list.
- `correctTag`: Tag to identify the correct items.
- `wrongTag`: Tag that identifies the wrong items.

Details

The tags are just command of the type "\Tag" that must be found somewhere that is not commented out inside the last item at the end of the tree structure. Usually you will want to use the tags that already identify the document items for this.

(For example, in the exam class, the tags `\choice` and `\CorrectChoice` could be used naturally, without having to introduce extra commands in the document)

Value

Data Frame. With the following columns

- `index`: Just an index running from 1 to `n`, where `n` is the number of rows
- **For each layer of depth in the document**: Four columns,
  - `name of section_original`: Contains an integer identifying the numbering of this section in the original layer, as identified by the naming convention
CreateRandomExams

- `<name of section command>_original` Contains an integer identifying the numbering of this item in the original section, as identified by the naming convention
- `<name of section>` Contains an integer identifying the numbering of this section in the current layer, as identified by the ordering of the document inputted on this function
- `<name of section command>` Contains an integer identifying the numbering of this item in the current section, as identified by the ordering of the document inputted on this function

**For the last layer of depth** 5 columns if the wrongTag is not NULL, 4 columns otherwise,

- `<name of section>_original` Contains an integer identifying the numbering of this section in the original layer, as identified by the naming convention
- `<name of section command>_original` Contains an integer identifying the numbering of this item in the original section, as identified by the naming convention
- `<name of section>` Contains an integer identifying the numbering of this section in the current layer, as identified by the ordering of the document inputted on this function
- `<correctTag>` Contains an integer identifying the numbering of this item in the current section, as identified by the ordering of the document inputted on this function
  - If the correctTag wasn’t found in this item, it will show NA instead. (This will only happen if wrongTag is not NULL, since otherwise this elements are omitted)
- `<wrongTag>` Contains an integer identifying the numbering of this item in the current section, as identified by the ordering of the document inputted on this function
  - If the wrongTag wasn’t found in this item, it will show NA instead. (This will only happen if wrongTag is not NULL, since otherwise this elements are omitted)

**See Also**

- FindExamAnswers for the exact underlying messy algorithm that controls how the table is created.
- Other Extracting information: CountNumberOfSections, FindExamAnswers, GenerateShortAnswerSheet

**Examples**

```r
ConstructAnswerSheet(
  TexExamRandomizer::testdoc$document,
  "CorrectChoice",
  "choice"
)
```

**Description**

This function creates a series of randomized exams from a tex document and personalizes the information from a table (if a table is given) and a series of command names where the information should be replaced.
Usage

CreateRandomExams(x, layersNames = c("questions", "choices"),
layersCmd = c("question", "\{choice\CorrectChoice\}"), outputBaseName,
outputDirectory, cmdReorder = rep_len(TRUE, length(layersNames)),
sectionReorder = FALSE, infoTable = NULL, colNames = NULL,
cmdNames = NULL, nOutputVersions = nrow(infoTable),
nOutputQuestions = "max", answerSheetCorrectTag = NULL,
answerSheetWrongTag = NULL, optionList = NULL)

Arguments

x
A character vector, each element represents one line of the LaTeX document

layersNames
A character vector, with each element representing the environment name to
be searched as cmdName as described in FindBegin and FindEnd

layersCmd
A character vector, with the same length as layersNames, with each element
representing the environment command to be searched as cmdName as described
in FindCommand.

outputBaseName
String, The basename for the output files.

outputDirectory
String, The output directory.

cmdReorder, sectionReorder
Logical vector, the length of cmdReorder determines how many layers deep are
we going to dig and randomize. For that reason, if sectionReorder is just a
scalar, it will assume that it repeats for every cmdReorder that is given. See
RandomizeDocument for extra details on these parameters.

infoTable
Table with information, if NULL, no information is added to the exams

colNames
Character vector, Column names from the infoTable from which we will ex-
tract the information.

It first tries to find the column names literally, if it couldn’t find them like that,
it will try to use them as a regular expression to find a column that matches
the column.

cmdNames
Character vector, Names of the commands on the tex file, \c{cmdNames[i]}, that
are to be matched with the columns to replace the information from the table in
those commands. For extra info see also ReplaceFromTable

nOutputVersions
Number of different random versions of the exam to be outputted

nOutputQuestions
Number of “questions” on the output exams. If the input is a scalar, the program
will decide how to more evenly split the questions between all the sections, oth-
wise one can directly provide an integer vector specifying how many questions
from each section are needed. (this only searches the “items” of the outermost
layer)

answerSheetCorrectTag, answerSheetWrongTag
If the tags are not given, the output answersheet will be NULL. In other cases,
these tags can be regular expressions
DivideFile

**optionList** Instead of writing the options on the function. Options could be given to optionList, and it will add those options. As long as the names are correct.

**Details**

All the output exams are named with `outputBaseName` followed by `00i` identifying the number of the exam (The number of zeros is the minimum that allows for all the exams to have a different number) and "._Version_." followed by the version number of the exam and ".tex". That is:

`<outputDirectory>/<outputBaseName>00i_Version_j.tex`

The number of exams outputted will always be the same as the number of versions if no table is given. However, if a table is added as input. It will create one exam for each row of the table, and it will try to divide as evenly as possible how to give the versions between the different rows. (Having one exam for each row, which will probably represent a student)

**Value**

A list that contains:

- `outputDirectory` The output directory
- `outputFiles` A character vector that contains all the output names
- `fullanswersheet` The full answer sheet of all the exams.

Each answer sheet is created as described by `ConstructAnswerSheet`, and all the answer sheets are joined together with a version number in front as an added column to bind them all together. The original version has the number 0, all the output versions have sequential numbers as Version.

This wrapper function assumes equal depth on all branches of the tree structure, so that the number of columns is always identical in the answer sheet.

**See Also**

- `ConstructAnswerSheet`, `ReplaceFromTable`, `RandomizeDocument` for extra details.
- To see examples of how to use it, look at the code in `jsonhwparser`.

---

**Description**

Function that takes a vector of text lines, `x`, and divides it in preamble and document.

**Usage**

`DivideFile(x)`

**Arguments**

- `x` A character vector, each element represents one line of the latex document
fun_from_folder

Details

It ignores everything after the first \end{document} and it will throw an error if it finds more than one \begin{document} before that.

Value

Returns a list with two character vectors:

- preamble A character vector that includes every line of \texttt{x} up to \begin{document}
- document A character vector that includes every line from \texttt{\begin{document}} to the first \end{document}

See Also


Examples

```r
file <- system.file(
  "extdata",
  "ExampleTexDocuments",
  "exam_testing_jsonparser.tex",
  package = "TexExamRandomizer"
)
x <- readLines(file)
DivideFile(x)
```

fun_from_folder  \hspace{1cm} Apply function within a folder

Description

It executes the function \texttt{fun} by first switching directories temporarily to the folder \texttt{folder} and then returning to the working directory.

Usage

```r
fun_from_folder(folder, fun, ...)
```

Arguments

- folder The folder of execution that the function is switched to before executing \texttt{fun}
- fun Function to be executed from the relative path
- ... Options to be passed to \texttt{fun}

Value

The return value of \texttt{fun(...)}
GenerateHomework

Examples

```r
list.files()
fun_from_folder(system.file("data", package = "TexExamRandomizer"), list.files)
list.files()
```

---

**GenerateHomework**

**Generate Homework**

**Description**

This function personalizes a 'LaTeX' document with data from a table, generating a new file for each row which is saved on the `outputDirectory`.

**Usage**

```r
GenerateHomework(x, Table, CommandNames, ColumnNames, outputDirectory, outputbasename)
```

**Arguments**

- **x** A character vector, each element represents one line of the latex document
- **Table** Data frame from which to extract the information
- **CommandNames** Character vector with the same length as `ColumnNames`
- **ColumnNames** Character vector with the names of the columns to be used
- **outputDirectory** The directory in which the output will be placed
- **outputbasename** The starting name for the output files
  The files will look like
  `<outputDirectory>/<outputbasename>_.00<number>.tex`
  Where the number of zeros is the minimum number of zeros required to have a different version number for each file. (i.e., if there is only 45 files, it is 01-45; but with 132 files, it would be 001-132)

**Details**

The command names should be 'LaTeX' commands that are being defined through

```latex
\texttt{\textbackslash newcommand\{\texttt{\textbackslash CommandNames[i]}\}\{\texttt{\textbackslash previous\ definition}\}}
```

The definition of these commands will be changed to be

```latex
\texttt{\textbackslash newcommand\{\texttt{\textbackslash CommandNames[i]}\}\{\texttt{\textbackslash Table[ColumnNames[i]][file #]}\}}
```

And it will output one file for each command.

The intent of this function was to populate information into a generic homework to personalize it for every student using 'LaTeX'. (It actually generalizes to maybe other problems).
GenerateShortAnswerSheet

Generating a short answer sheet

Description

Given a number of answer sheets generated by ConstructAnswerSheet that have been binded together. And that have a column, versionColName, that identifies each version. It collects all the answers together and places all the answers together for each exam.

Usage

GenerateShortAnswerSheet(ExamSheet, versionColName = "Version",
correctColName = "CorrectChoice")

Arguments

ExamSheet a exam sheet that contains all versions, similar to CreateRandomExams
versionColName The name of the column in the original exam that contains the version number
correctColName The name of the column that contains the last index for the correct tag, or NA if it is not a correct choice.

Details

Note that if the version number is 0, it is ignored, since it understands that version 0 is the reference version.
If the document has more than two layers, keep in mind that it just shows the top most layer numbering and then the inner most number of the correct answers.
Note how this implies as well that an exam with more than one possible answer can not be simplified into a short answer sheet.
IMPORTANTLY, If a certain exam has less answers than other exams, the are just cited sequentially. Which may cause confusion. To clarify. This may happen if a certain question has more than one solution marked as "correct", or if a certain question has no solutions marked as correct. In that case, The short answer sheet just sequentially names all the correct answers, disregarding which questions they are referring to. (This is a very special case that will only come up in a real scenario if you are writing a short answer question in the middle of a multiple choice test. Or if you are writing some questions to have multiple correct answers, but only a few of them, and those questions are not included in all exams... (So evil))
GradeExams

Value
A data frame

• Each row identifies one version of the answer sheet
• the first column is the version number, the rest of the columns are the questions.

See Also
Other Extracting information: ConstructAnswerSheet, CountNumberOfSections, FindExamAnswers

Examples

csvfile <- system.file(
  "extdata",
  "ExampleTables",
  "ExampleAnswerSheet.csv",
  package = "TexExamRandomizer"
)
testASheet <- read.csv(
csvfile,
header = TRUE,
stringsAsFactors = FALSE,
na.strings = c("", "NA", "Na"),
strip.white = TRUE
)
GenerateShortAnswerSheet(testASheet)

GradeExams  GradeExams

Description
Grades an exam given a parsed list by WhichAnswerOriginal

Usage
GradeExams(ExamAnswerParsedList, name.ColCorrect, name.ColIncorrect,
MaxOutputGrade = 100, ExtraPoints = 0, ExtraPointsForAll = 0)

Arguments
ExamAnswerParsedList
List parsed by WhichAnswerOriginal
name.ColCorrect, name.ColIncorrect
The names of the correct and incorrect columns in each answer sheet of the ExamAnswerParsedList respectively.
MaxOutputGrade  Maximum score that one should get if you get a perfect score, before counting the ExtraPoints.

ExtraPoints  Extra points to be added after scoring the exam. This points are added after the scaling is done with MaxOutputGrade.

ExtraPointsForAll  Scalar numeric value, extra points to be given to all student.

Details

The score is first added on the base of the number of questions that are found on every parsed list. If a question is removed from an exam, not all students may have that question as explained in the "Removing questions from the exam" section. If the total rows of a certain student list is $n$, the score is

$$c/n \times \text{MaxOutputGrade}$$

, where $c$ is the number of correct answers.

After that is done, the ExtraPoints are added.

Value

It returns the StudentInfo attribute of the parsed list adding the following columns to it:

$\text{addedPoints}$ Individual part of ExtraPoints

$\text{addedAllPoints}$ Extra Points For All

$\text{maxGrade}$ Max number of questions for the exam. (It would be different if when removing a question, some students didn’t have a question in that exam)

$\text{Grade}$ Number of correct answers that a student wrote in an exam

$\text{Grade_TotaledExam}$ This is the total grade as explained on the Extra Points section.

Extra Points

The structure of ExtraPoints and the convention on how the score is calculated taking it into account is worth mentioning in its own section. The score is calculated as:

$$total_{grade} = (c + \text{extra}_{all})/(\text{max}_{n} + \text{extra}_{all}) \times \text{MaxOutputGrade} + \text{extra}_{individual}$$

Where

$c$ Number of correct questions

\text{extra}_{all} Number of extra points for all.

This is thought of to be used as a question that you removed from the exam last minute, but that you want to actually count it as correct for every single student. I.e., a question that everyone got correct but it is not taken into consideration in the grading.

\text{extra}_{individual} Number of extra points for that student.

\text{max}_{n} Maximum number of questions in the students exam, which may differ from other students if you had to removed a bugged questions that not everyone had

MaxOutputGrade The scaling to be done. This should be the maximum grade any student "should" get. (The individual extra points are added after the scaling is done)
Removing Questions from the exam

Note that if after creating the exam, you found that a question is bugged and can’t be used to grade the exam, all you have to do is tell the student to answer “something” and you only have to remove it from the original/reference version in the Full Answer Sheet. When you apply the grading function, that question will then be ignored.

Notice how this creates output lists with different lengths in the case that two students didn’t have that same question in their exam.

For example, if a exam has 15 questions out of a 50 question document. If student A has a bugged question and student B doesn’t, the answer sheet produced for student A will have 14 rows while the one for student B will have 15 rows.

See Also

Other Grading Exams: ObtainExamStats

Examples

#First part coming from FindMatchingRow example

asheet_file <-
system.file(
  "extdata",
  "ExampleTables",
  "ExampleAnswerSheet.csv",
  package = "TexExamRandomizer")
responses_file <-
system.file(
  "extdata",
  "ExampleTables",
  "ExampleResponses.csv",
  package = "TexExamRandomizer")
FullAnswerSheet <-
read.csv(
  asheet_file,
  header = TRUE,
  stringsAsFactors = FALSE,
  na.strings = c("", "NA", "Na"),
  strip.white = TRUE)
Responses <- read.csv(
  responses_file,
  header = TRUE,
  stringsAsFactors = FALSE,
  na.strings = c("", "NA", "Na"),
  strip.white = TRUE)
compiledanswers <-
WhichAnswerOriginal(
  StudentAnswers = Responses,
  FullExamAnswerSheet = FullAnswerSheet,
  names.StudentAnswerQCols = grep(
    names(Responses),
    )
pattern = "^\d.*[[:digit:]]$",  
value = TRUE),
names.StudentAnswerExamVersion = grep(
  names(Responses),
  pattern = "Version",
  value = TRUE),
OriginalExamVersion = 0,
names.FullExamVersion = "Version",
names.FullExamOriginalCols = grep(
  names(FullAnswerSheet),
  pattern = "_original",
  value = TRUE),
names.CorrectAndIncorrectCols = c(
  "choice",
  "CorrectChoice")
}
# Actual Code

ExtraPoints_individual <- runif(nrow(Responses), min = 1, max = 10)
ExtraPoints_forall <- 2
GradedStudentTable <-
  GradeExams(
    compiledanswers,
    name.ColCorrect = "CorrectChoice",
    name.ColIncorrect = "choice",
    MaxOutputGrade = 100,
    ExtraPoints = ExtraPoints_individual,
    ExtraPointsForAll = ExtraPoints_forall
  )

#jsonexamparser

Json Exam Document Parser

Description

This function takes a series of options as obtained from parse_args through the parameter opt. The "examples" section provides all the options that it can parse.

From within those options, a --file option is mandatory.

The file option provides a 'LaTeX' file name in which to search for lines on the preamble \%!TexExamRandomizer within the first 200 lines.

With those options that it finds through tags, it passes the function CreateRandomExams.

Note that the tags must respect the JSON format, that is. It needs to be written within double quotes.
Usage

jsonexamparser(opt)

Arguments

opt Options as parsed from parse_args. The function expects a series of options, the example code exemplifies those options that the function understands.

Details

All the options can be found on

vignette("ExamOptions", package = "TexExamRandomizer")

The options that are called "command line" options in the vignette are those that are given to the function through opt, the rest of the options are read directly from the document specified with --file <filename>

See Also

Other jsoncompiler: ParsePreambleForOptions, jsonhwparsers

Examples

```r
## Not run: 
#!/bin/Rscript 
#This example showcases the type of script this jsonparser might be used on. 
# You can still use it without a script, 
# just by adding a list that has the same names as the list provided in opt 
library(optparse)
option_list <- list(
  make_option( 
    c("--file"),
    action = "store",
    default = NULL,
    type = 'character',
    help = "Filename of the Tex File"
  ),
  make_option( 
    c("--table"),
    action = "store",
    default = NULL,
    type = 'character',
    help = "Filename of the table to break down. It overwrites the values written on the file"
  ),
  make_option( 
    c("-n", "--noutput"),
    action = "store",
    default = NULL,
    type = "integer",
    help = "Number of output Versions"
  ),
  make_option( 
```
c("-q", "--nquestions"),
action = "store",
default = NULL,
type = "character",
help = "Number of output questions"
),
make_option(
  c("-s", "--seed"),
  action = "store",
default = NULL,
type = "integer",
help = "Seed for any randomization done"
),
make_option(
  c("-c", "--compile"),
  action = "store_true",
default = FALSE,
type = "logical",
help = "Should the output folder be compiled or not"
),
make_option(
  c("--xelatex"),
  action = "store_true",
default = FALSE,
type = "logical",
help = "Should we use xelatex to compile or not"
),
make_option(
  c("-d", "--debug"),
  action = "store_true",
default = FALSE,
type = "logical",
help = "If debugging, it doesn't remove auxiliary files"
)
)

### PARSING OPTIONS ###

####

opt <-
  parse_args(
    OptionParser(option_list = option_list),
    positional_arguments = TRUE
  )

# Let's assume the file was the example file
testfile <-
system.file(
  "extdata",
  "ExampleTexDocuments",
  "exam_testing-nquestions.tex", # Test exam that doesn't require a table
  package = "TexExamRandomizer")
# To prevent modifying the file system in examples
temporalfile <- paste(tempfile(), ".tex", sep = "")

file.copy(testfile, temporalfile)

opt$options$file <- temporalfile

jsonexamparser(opt)

## End(Not run)

---

**Json Homework Parser**

**Description**

This function takes a series of options as obtained from `parse_args` through the parameter `opt`. The "examples" section provides all the options that it can parse.

From within those options, a `--file` option is mandatory.

The file option provides a 'LaTeX' file name in which to search for lines on the preamble `\LaTeXexamrandomizer` within the first 200 lines.

With those options that it finds through tags, it passes the function `GenerateHomework`.

Note that the tags must respect the JSON format, that is, it needs to be written within double quotes.

**Usage**

```
jsonhwparsar(opt)
```

**Arguments**

- `opt` Options as parsed from `parse_args`. The function expects a series of options, the example code exemplifies those options that the function understands.

**Details**

It acts similarly to `link(jsonexamparser)`, but with the exception of not providing any randomization option, it only provides the personalization options.

Look at `vignette("ExamOptions", package = "TexExamRandomizer")` to see the details of the options that it accepts.

**See Also**

Other jsoncompiler: `ParsePreambleForOptions, jsonexamparser`
Examples

```r

## Not run:
#!/bin/Rscript
# This example showcases the type of script this jsonparser might be used on.
# You can still use it without a script, just by adding a list that has the same names as the list provided in opt
library(optparse)
option_list <- list(
  make_option(
    c("--file"),
    action = "store",
    default = NULL,
    type = "character",
    help = "Filename of the Tex File"
  ),
  make_option(
    c("--table"),
    action = "store",
    default = NULL,
    type = "character",
    help = "Filename of the table to break down. It overwrites the values written on the file"
  ),
  make_option(
    c("--seed"),
    action = "store",
    default = NULL,
    type = "integer",
    help = "Seed for any randomization done"
  ),
  make_option(
    c("--compile"),
    action = "store_true",
    default = FALSE,
    type = "logical",
    help = "Should the output folder be compiled or not"
  ),
  make_option(
    c("--xelatex"),
    action = "store_true",
    default = FALSE,
    type = "logical",
    help = "Should we use xelatex to compile or not"
  ),
  make_option(
    c("--debug"),
    action = "store_true",
    default = FALSE,
    type = "logical",
    help = "If debugging, it doesn't remove auxiliary files"
  )
)
```

### PARSING OPTIONS ###

```r
opt <-
  parse_args(
    OptionParser(option_list = option_list),
    positional_arguments = TRUE
  )

# Let's assume the file was the example file
testfile <-
  system.file(
    "extdata",
    "ExampleTexDocuments",
    "exam_testing_nquestions.tex", #Test exam that doesn't require a table
    package = "TexExamRandomizer"
  )

# To prevent modifying the file system in examples
temporalfile <- paste(tempfile(), ".tex", sep = "")

file.copy(testfile, temporalfile)

opt$options$file <- temporalfile
```

```r
jsonhwparsers(opt)
```

```r
## End(Not run)
```

---

**ObtainExamStats**  

*Obtaining exam statistics*

---

**Description**

This function gets an answer sheet of the original version of the exam as a data frame, and a parsed list, which is obtained from GradeExams and it outputs the statistics of how many answers are parsed exam, that is graded and obtains from there.

**Usage**

```r
ObtainExamStats(OriginalExamAnswerSheet, ExamAnswerParsedList, names.FullExamOriginalCols)
```

**Arguments**

- **OriginalExamAnswerSheet**  
  The answer sheet of the original exam. (In this package the convention is the exam version "0")
ExamAnswerParsedList

A parsed list for every student, as outputted by GradeExams

names.FullExamOriginalCols

Names of those columns that in the answer sheet identify for all versions where that item is found on the original columns, (i.e., as ordered from the original version exam)

Value

Returns the OriginalExamAnswerSheet with a column added to it, named "ExamAnswerCount" that counts the number of answers for each question

See Also

Other Grading Exams: GradeExams

Examples

asheet_file <-
system.file(
    "extdata",
    "ExampleTables",
    "ExampleAnswerSheet.csv",
    package = "TexExamRandomizer")
responses_file <-
system.file(
    "extdata",
    "ExampleTables",
    "ExampleResponses.csv",
    package = "TexExamRandomizer")
FullAnswerSheet <-
read.csv(
    asheet_file,
    header = TRUE,
    stringsAsFactors = FALSE,
    na.strings = c("", "NA", "Na"),
    strip.white = TRUE)
Responses <- read.csv(
    responses_file,
    header = TRUE,
    stringsAsFactors = FALSE,
    na.strings = c("", "NA", "Na"),
    strip.white = TRUE)
compiledanswers <-
WhichAnswerOriginal(
    StudentAnswers = Responses,
    FullExamAnswerSheet = FullAnswerSheet,
    names.StudentAnswerQCols = grep(
        names(Responses),
        pattern = "^Q.*\[[[:digit:]]\]",
        value = TRUE),
ParsePreambleForOptions

Description

This function parses a preamble of a document trying to read options handed to the package TexExamRandomizer to be used in compiling.

Usage

ParsePreambleForOptions(preamble)

Arguments

preamble character vector identifying the preamble from which to pass the JSON readon through

Details

It finds all \texttt{\%!TexExamRandomizer = \{\}} lines. It then uses the function \texttt{fromJSON} to parse them, and it concatenates all those options.

If more than one option with the same name is given, it tries to concatenate those. However, it doesn’t do that recursively, only if the names of the outer layer are the same... therefore, in nested
structure you might end up with a list that have twice the same name. Keep in mind that in those cases, the default behaviour of R is to select the first one.

Value

Returns a list, that concatenates all the lists of options described on the file.

See Also

Other jsoncompiler: jsonexamparser, jsonhwpaser

Description

Function to randomize a Document, as created by `StructureDocument`.

It Randomizes each layer according to the prescriptions involved in the internal function `GetLayerSampleIndexes`. Which, in summary, randomizes each section inside, and then randomizes the orders of the sections.

**Important note:** One must provide to this function the document part of the structure. Since `StructureDocument` provides as the outer most layer a split between the preamble and the document, one must just supply the document part to this function, (or a subsection of it).

Usage

`RandomizeDocument(Document, isSectionReordered.vector, isLayerRandomized.vector)`

Arguments

- **Document**  
  Document to randomize, as generated by `StructureDocument`. The names of the structure are used to determine how to randomize the document.

- **isSectionReordered.vector**  
  Logical vector, specifying if the order of sections should be also randomized at a certain depth level.

  **Note** that if `isLayerRandomized` is set to false for a certain layer, `isSectionReordered` will have no effect. (Probably this isn’t the best behaviour)

- **isLayerRandomized.vector**  
  Logical vector, specifying if you should randomize the order of the items, (denoted by `\cmdName`) or not at a certain depth level.

  This vector should have the same length as the depth at most, otherwise it will raise an error if you try to "dig deeper than it can". And `isSectionReordered.vector` should have matching elements for each element of `isLayerRandomized.vector`

  (Maybe we could change this to a warning instead? To allow for structures with different depths within different branches of the tree)
RandomizeDocument

Details

It keeps randomizing recursively inner layers of the structure until it runs out of elements on the logical vectors isSectionReordered.vector and isLayerRandomized.vector.

A "section" denotes the content within a begin-end environment in the document. Each section is then assumed to be divided in a beginning and end parts, that should be fixed in place, and the parts denoted by the command \cmdName as explained on StructureDocument.

We will denote those parts as "items." Analogously to itemize environments in 'LaTeX'.

The purpose of this function is therefore to randomize the items from the structure, fixing the begin and end parts within a section. And then to reorder each section while keeping the pre- and post-parts fixed, and to do so recursively until we exhaust the isLayerRandomized.vector

isSectionReordered.vector specifies whether to order sections for a certain depth, while isLayerRandomized.vector specifies whether to order the items within a section of that same depth.

In some cases you may want to reorder the sections, for example, using the examdesign class. Over there, questions use the begin-end question format.

In others cases you may want to preserve the order of sections while still modifying the order of the items, like when you are using the exam class, or when creating your own list of questions with an \itemize environment.

For efficiency, if you don’t want to randomize to the full depth of your tree, just make those logical vectors of your desired length, rather than making them of length $n$ and then setting every layer after the last one you want to randomize to false. That will prevent the program from walking down the whole tree checking everything.

Value

A document structure, as provided by StructureDocument.

However, the names of the structure will no longer be sequential, the naming convention in the new structure will refer to the original structure that was inputted into this function. Which is very useful when you want to keep track of where things have moved.

See Also

StructureDocument, TODO: Add reference to extracting info functions

Examples

rndDoc <- RandomizeDocument(
    TexExamRandomizer::testdoc$document,
    c(FALSE,TRUE),
    c(TRUE,TRUE)
)
Description

Given a \LaTeX\ file represented as a character vector with \texttt{x}, it replaces from a table the commands given by \texttt{commandNames} for the values found on the table.

\\texttt{\newcommand{\commandName[i]}{table[tableRow, columnName[i]]}}.

Usage

\texttt{ReplaceFromTable(x, table, tableRow, columnNames, commandNames)}

Arguments

\begin{itemize}
\item \texttt{x} \hfill A character vector, each element is suppose to represent a line
\item \texttt{table} \hfill Data frame from which to extract the information
\item \texttt{tableRow} \hfill Integer, row of the table to be used
\item \texttt{columnNames} \hfill Character vector with the names of the columns to be used
\item \texttt{commandNames} \hfill Character vector with the same length as \texttt{columnNames}. Contains the names of the \LaTeX\ commands to be replaced.
\end{itemize}

Details

To do the replacement for each item, it uses the function \texttt{ReplacePreambleCommand}. See the details in that function for more information.

Value

A character vector, representing the text \texttt{x}, where all instances of \texttt{\newcommand{\commandName[i]}{<random text>}} have been replaced with \texttt{\newcommand{\commandName[i]}{table[tableRow, columnName[i]]}}.

See Also

Other Preamble adjustment: \texttt{ReplacePreambleCommand}

Examples

\begin{verbatim}
custom_preambles <- list()
for (i in 1:nrow(TexExamRandomizer::testclass)) {
custom_preambles <- c(custom_preambles, 
list(TexExamRandomizer::ReplaceFromTable(TexExamRandomizer::testdoc$preamble,
\end{verbatim}
ReplacePreambleCommand

Description

This function gets a character vector in which each element represents a line of a preamble of a 'LaTeX' document, and it replaces the definition of the command \commandName to have the value commandValue.

Usage

ReplacePreambleCommand(x, commandName, commandValue)

Arguments

x A character vector, each element is suppose to represent a line
commandName A string identifying either the command name
commandValue Replacement for the definition of commandName

Details

It only modifies the value of the command by replacing instances of 
\newcommand{\commandName}{<previous definition>} with instances of 
\newcommand{\commandName}{<commandValue>}. Keep in mind that both commandName and commandValue are placed directly inside a regex.

If you want to "hide" a certain definition of a command from being found and replaced by this function, simply define it by using \def or \newcommand* or a \renewcommand when you define them.

Make sure you are using a one-line definition in commands that you want replaced, since this won’t be able to detect commands that are defined in multiple lines in 'LaTeX'.

Also, note how certain invalid things in 'LaTeX' would still be matched by this regex, however you should find those errors before you start using this program since those errors would not allow you to compile the 'LaTeX' document on the first place.

Lastly, if it doesn’t find a command on the document, it silently ignores it.
StructureDocument

Value
A character vector, with the preamble, replacing all instances of!*\newcommand\commandName{<random text>!} with!*\newcommand\commandName{commandValue!}

See Also
Other Preamble adjustment: ReplaceFromTable

Examples
new_preamble <- ReplacePreambleCommand(TexExamRandomizer::testdoc$preamble, "nickname", "Alex")

Description
Function that takes a character vector, *x*, representing a 'LaTeX' file and it outputs a tree structure with the structure specified by *layersNames* and *layersCmd*.

It assumes *x* is representing a 'LaTeX' file that can has been checked it compiles apropitaly before we make any modification.

Note however that this function only moves lines around, it doesn’t split a line in two.

Usage
StructureDocument(*x*, layersNames, layersCmd)

Arguments
*x* A character vector, each element represents one line of the latex document
*layersNames* A character vector, with each element representing the environment name to be searched as cmdName as describe in FindBegin and FindEnd
*layersCmd* A character vector, with the same length as *layersNames*. with each element representing the environment command to be serached as cmdName as described in FindCommand.

Details
Both *layersNames* and *layersCmd* must have the same length, since for each index, *i*, *layersNames[i]* and *layersCmd[i]* refer to one layer of the tree structure of the document. Consequent layers must be found inside previous layers.

If it finds the structure of the document to not be completed, it will throw an error.
Value

It returns a list, with each element having a name. Recreating the tree structure identified by `layersNames` and `layersCmd` in the text file `x`.

It first divides the document into two lists:

- **preamble** Contains a character vector identifying everything before the `\begin{document}`
- **document** Contains the tree structure identifying the document

Now, the naming convention for each layer of the document is as follows. We will use the convention `<layerName>`, `<layerCmd>`.

Note the convention first, everything that it finds prior to the first environment, it throws it into a character vector that it calls `prior_to_<layerName>`. After the first environment `<layerName>` ends, it assumes that everything from that `\end{<layerName>}` onwards corresponding to the next environment, and it will throw it to the prior part of that one, `post_to_<layerName>`

- **prior_to_<layerName>** Includes everything up to the first `\begin{<layerName>}` without including that line
- **1_<layerName>_begin_<layerName>** Includes the `\begin{<layerName>` for the 1st section, and everything until it finds the first `\<layerCmd>`
- **1_<layerName>_1_<layerCmd>** Includes everything from the 1st `\<layerCmd>` until the second `\<layerCmd>`, without including the line in which the second command is found
- **1_<layerName>_2_<layerCmd>** Same thing... and it keeps going until the last `\<layerCmd>` is found
- **1_<layerName>_end_<layerName>** It includes the `\end{<layerName>}` for the 1st section.

... It then repeats the same structure for the next environment, changing the naming convention to start with `2_<...>` and so on until it does the last environment

- **post_to_<layerName>** After the last layer ends with `\end{<layerName>}`, it throws the rest of the lines into this last character vector

This structure is applied recursively to each `i_<layerName>_j_<layerCmd>` of the previous layer to find the structure for the next layer. The result is a tree of lists, with names that identify the whole structure, and the ending node of each branch is always a character vector

**IMPORTANT NOTE:** Note that this function only rearranges the lines of the document, it can’t split a document between a line. So if you want to make sure something always stays together, put them both in the same line. This is intentional, to force a more clear structure on the document that will be parsed

In Summary, the sketch of the tree structure would be:

- **preamble**
- **Document**
  - **prior_to_LayerName[1]**
  - **1_LayerName[1]_begin_LayerName[1]**
  - **1_LayerName[1]_1_LayerCmd[1]**
    * **prior_to_LayerName[2]**
    * **1_LayerName[2]_begin_LayerName[2]**
testclass

* 1\_layerName[2]_1\_layerCmd[2]
  · Continues...
* 1\_layerName[2]_2\_layerCmd[2]
  · Continues...
* ...
* post\_to\_layerName[2]
  – 2\_layerName[1]_begin\_layerName[1]
  – 2\_layerName[1]_1\_layerCmd[1]
  * ...
  – ...
  – n\_layerName[1]_end\_layerName[1]
  – post\_to\_layerName[1]

If a \langle layerCmd\rangle is not found inside an environment, everything inside that environment is thrown into the begin\_layerName part and instead of the numbered environments, an empty character list is added in the middle, with name empty\_<layerCmd> section.

See Also

FindStructure for more information on the details of how the layers are found.

Other Structuring Document: CompileDocument, DivideFile, FindStructure, IsWellSectioned

Examples

```r
file <- system.file(
  "extdata",
  "ExampleTexDocuments",
  "exam_testing_jsonparser.tex",
  package = "TexExamRandomizer"
)
x <- readLines(file)
layersNames <- c("questions", "choices")
layersCmd <- c("question", "(choice|CorrectChoice)"
doc <- StructureDocument(x, layersNames, layersCmd)
```

<table>
<thead>
<tr>
<th>testclass</th>
<th>Sample class table</th>
</tr>
</thead>
</table>

Description

Sample class for testing with five students. The variables stored for each student are as follows

Usage

testclass
Format

A dataframe with 5 rows and 4 columns

Details

• Class
• Roll.Number
• Nickname
• Name

Source

self

Description

A simple sample TeX document to test the package easily before deploying solutions.

Usage

testdoc

Format

A list with the format described in StructureDocument

Source

Created between me and my students in Suankularbwittayalai Rangsit School
WhichAnswerOriginal

Description

Given the answers of the students gathered in a table, and a full answer sheet of all versions (Including a "reference/original" version), it finds where those answers are found in the original exam, by copying from the original version the matching rows and binding them in order for every student. It then combines all of them in a list, and includes as well all the remaining student information in the attribute "StudentInfo".

It is intended as an internal function to generate the grades, and to identify in a very general way where the answers of the students are (relative to the reference/original version).

Usage

WhichAnswerOriginal(StudentAnswers, FullExamAnswerSheet, OriginalExamVersion = 0, names.FullexamVersion = "Version", names.FullexamOriginalCols, names.CorrectAndIncorrectCols, names.StudentAnswerQCols, names.StudentAnswerExamVersion)

Arguments

StudentAnswers DataFrame, each row is a student, each column is some information about said student. Any column not included in names.StudentAnswerQCols will be understood as information of the student and will be saved as part of the information table when we output the result.

FullExamAnswerSheet Answer sheet of all the exam versions, following the conventions of the FullAnswerSheet outputted by CreateRandomExams

OriginalExamVersion The version of the original exam, without randomization, as stored on the FullExamAnswerSheet. The default value is 0, as that is the convention on CreateRandomExams

names.FullexamVersion The name of the column in which the version of the exam is stored on the FullExamAnswerSheet. The default value is "Version", as that is the convention on CreateRandomExams

names.FullexamOriginalCols The names of the columns that contain the information of the items relative to where they were positioned in the original ordering of the exam, before randomizing the exam. The convention from CreateRandomExams is to finish all of them by ".original"

names.CorrectAndIncorrectCols It should be a character vector. The names of the columns in the FullExamAnswerSheet that contain the correct and incorrect answers, in that order. This column should have an integer value if it is indeed a correct value in the correct column and a incorrect value in the incorrect value, and NA otherwise. (The should be "complementary")
names.StudentAnswerQCols
The names in the StudentAnswers that store the answers from every student to the exam, ordered. These columns should contain integers values. Where 1 refers to the first answer, and n refers to the nth answers in their exam.

names.StudentAnswerExamVersion
The name of the column in the StudentAnswers that identifies the version of the exam.

Details

The StudentAnswers should be a data frame with one student answers represented by every row. The answers of the student to the exam should be ordered.

It is important that the columns named names.StudentAnswerQCols should contain all their answers, if a student didn’t answer a question leave a NA or an invalid integer value as an answer, like 0, or a number larger than the number of answers to that question, so that is is found as out of bounds.

Value

It returns a list. Each element of the list is a dataframe, and there is one dataframe for each student in the StudentInfo table provided.

All the columns that are not in the columns names.StudentAnswerQCols are regarded as "StudentInfo", and they are added to the attribute "StudentInfo" of the output as a data frame.

List elements: They are outputted in order, that is to say, for StudentAnswers[i,] the list that provides the information for that row will be outputlist[[i]].

outputlist[[i]] is a dataframe that identifies the rows that the student answered as they are found on the original/reference version. Therefore, if a student answeres a certain value, and that value is not reflected on the original version, it get's ignored.

StudentInfo attribute A dataframe containing all the student information that wasn’t their answers.

Underlying algorithm

To identify the rows on the original exam it does the following:

1. It first finds their exam in the full answer sheet by their exam version.
2. After that, it removes from their exam the rows that identify the correct/incorrect choices.
3. By trying to match that row with a row on the reference exam it can tell where that question is found on the original exam.
4. Then it identifies where that question is found on the original version, and it finds there which of the possible correct/incorrect choices is found.
5. If it didn’t find any correct/incorrect choice matching the value given by the student, it marks it as out of bounds and replaces both correct and incorrect columns with NA.
6. If it still doesn’t find the row, it simply ignores it, and the output will have one less row.
7. Now you can tell how many questions the student answered correctly by looking at how many values are not NA in the correct choice column of the output list.
Removing Questions from the exam

Note that if after creating the exam, you found that a question is bugged and can’t be used to grade the exam, all you have to do is tell the student to answer “something” and you only have to remove it from the original/reference version in the Full Answer Sheet. When you apply the grading function, that question will then be ignored.

Notice how this creates output lists with different lengths in the case that two students didn’t have that same question in their exam.

For example, if a exam has 15 questions out of a 50 question document. If student A has a bugged question and student B doesn’t, the answer sheet produced for student A will have 14 rows while the one for student B will have 15 rows.

Notes

Note1: Remember that in the original answer sheet there are two columns, one with correct choice, another one with wrong choice. If the value is NA of one of those two columns it SHOULD NOT be NA on the other row.

Note2: The idea is that the data frames can be read to know the score of the student by counting the number of values that are not NAs on the correct choice column. (The numbers on the correct/incorrect columns themselves can be used for statistical purposes, to tell how many students answered each question).

Note3: The data frames can be used for many other statistical purposes very easily.

See Also

GradeExams and ObtainExamStats for examples on how to use the output of this function to obtain more detailed information.

Examples

```r
asheet_file <-
  system.file(
    "extdata",
    "ExampleTables",
    "ExampleAnswerSheet.csv",
    package = "TexExamRandomizer")
responses_file <-
  system.file(
    "extdata",
    "ExampleTables",
    "ExampleResponses.csv",
    package = "TexExamRandomizer")
FullAnswerSheet <-
  read.csv(
    asheet_file,
    header = TRUE,
    stringsAsFactors = FALSE,
    na.strings = c("", "NA", "Na"),
```
WhichAnswerOriginal

```r
strip.white = TRUE
Responses <- read.csv(
  responses_file,
  header = TRUE,
  stringsAsFactors = FALSE,
  na.strings = c("" , "NA", "Na"),
  strip.white = TRUE)
compiledanswers <-
  WhichAnswerOriginal(
    StudentAnswers = Responses,
    FullExamAnswerSheet = FullAnswerSheet,
    names.StudentAnswerQCols = grep(
      names(Responses),
      pattern = "^Q\.*[[:digit:]]\$",
      value = TRUE),
    names.StudentAnswerExamVersion = grep(
      names(Responses),
      pattern = "Version",
      value = TRUE),
    OriginalExamVersion = 0,
    names.FullExamVersion = "Version",
    names.FullExamOriginalCols = grep(
      names(FullAnswerSheet),
      pattern = "_original",
      value = TRUE),
    names.CorrectAndIncorrectCols = c(
      "choice",
      "CorrectChoice")
  )
nicknames <- attr(compiledanswers, "StudentInfo")$Nickname

for (i in 1:length(compiledanswers)) {
  cat("Student\t" , nicknames[i] , " got\t",
       sum(is.na(compiledanswers[[i]]$CorrectChoice)),
       " questions correctly\n", sep = "")
}
```
Index

*Topic datasets
  testclass, 28
testdoc, 29
cat, 3
catDocument, 3
CompileDocument, 8, 28
ConstructAnswerSheet, 4, 7, 10, 11
CountNumberOfSections, 5, 11
CreateRandomExams, 2, 5, 10, 14, 30

DivideFile, 7, 28

FindBegin, 6, 26
FindCommand, 6, 26
FindEnd, 6, 26
FindExamAnswers, 5, 11
FindStructure, 8, 28
fromJson, 21
fun_from_folder, 8

GenerateHomework, 2, 9, 17
GenerateShortAnswerSheet, 5, 10
GetLayerSampleIndexes, 22
GradeExams, 11, 19, 20, 32

IsWellSectioned, 8, 28

jsonexamparser, 14, 17, 22
jsonhwparser, 7, 10, 15, 17, 22

ObtainExamStats, 13, 19, 32

parse_args, 14, 15, 17
ParsePreambleForOptions, 15, 17, 21

RandomizeDocument, 6, 7, 22
ReplaceFromTable, 6, 7, 10, 24, 26
ReplacePreambleCommand, 24, 25

StructureDocument, 3, 4, 8, 22, 23, 26, 29
testclass, 28
testdoc, 29
TexExamRandomizer
  (TexExamRandomizer-package), 2
TexExamRandomizer-package, 2
WhichAnswerOriginal, 11, 30