Package 'quickcode'

December 3, 2023

Type Package

Title A Compilation of Some Frequently Used R Functions

Version 0.6

Maintainer Obinna Obianom <idonshayo@gmail.com>

Description The NOT functions and a simple compilation of various functions for easy usage. Shorthand code to save memory usage.

License MIT + file LICENSE

URL https://quickcode.obi.obianom.com

BugReports https://github.com/oobianom/quickcode

Depends R (>= 3.6)

Imports utils, grDevices, stats, rstudioapi, tools, Polychrome

Suggests rmarkdown, knitr, qpdf, testthat

Encoding UTF-8

VignetteBuilder knitr

Language en-US

LazyData false

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Obinna Obianom [aut, cre],
Brice Richard [aut]

Repository CRAN

Date/Publication 2023-12-03 18:00:02 UTC

R topics documented:

`add.header` *Addin snippet function to add header comment to a current opened file*

Description

Shorthand to add header comment

Usage

```
add.header()
```

Value

Inserts header content for file

Examples

```
if(interactive())
add.header()
```

`add.sect.comment` *Addin snippet function to custom section comment*

Description

Shorthand to add section comment to current file

Usage

```
add.sect.comment()
```

Value

Inserts section comment content for file

Examples

```
if(interactive())
add.sect.comment()
```

`add.snippet.clear` *Snippet R function to clear console and set directory*

Description

Shorthand to add clear console code to current file

Usage

```
add.snippet.clear()
```

Value

Inserts code to clear console

Examples

```
if(interactive())
add.snippet.clear()
```

`add_key` *Add index keys to a vector or data frame or list or matrix*

Description

Index a vector or lists and convert to a list of objects

Usage

```
add_key(vector)
```

Arguments

`vector`	vector or data frame to transform

Details

This function takes a vector and turns it into a list containing 'key' and 'value' for each vector. This allows the output to be used in loops such as for loops or lapply or other functions to track the index of the list content e.g. 1,2,3...

This function also contains a validator to ensure that a vector had not been previously 'keyed', which prevents the user from inadvertently calling the function twice on a vector. Helps especially because the function keys the vector, and sets the new list to the variable name of the original vector.

Value

a transformed list containing keys along with vector values

Use case

Efficient for loops and for tracking various steps through a vector contents

Examples

```
#ex1 simple conversion of a vector
rti2 <- c("rpkg","obinna", "obianom")
add_key(rti2)
rti2

#ex2 add keys to a vector content for use in downstream processes
ver1 <- c("Test 1","Test 2","Test 3")
add_key(ver1)

#ex3 use keyed ver1 in for loop
for(i in ver1){
  message(sprintf("%s is the key for this %s", i$key, i$value))
}

#ex4 use keyed ver1 in lapply loop
xl1 <- lapply(ver1,function(i){
  message(sprintf("lapply - %s is the key for this %s", i$key, i$value))
})
```

`ai.duplicate` *Prompt guided duplication if files*

Description

AI like duplication and editing of files

Usage

```
ai.duplicate(file = NULL, new.name = NULL, open = TRUE)
```

Arguments

`file`	file to duplicate
`new.name`	OPTIONAL.name of new file
`open`	open file after duplication

Value

duplicated files with edited texts

Examples

```
if(interactive()){
file1s <- paste0(tempfile(),".R")
writeLines("message(
'Sample items: farm, shinyappstore, rpkg'
)", file1s)
ai.duplicate(file1s,'file2.R')
}
```

---

archivedPkg *Listing of all CRAN archived R packages*

Description

Retrieve a list of all currently archived R packages and their archive date

Usage

```
archivedPkg(
  startsWith = c("all", letters),
  after = NULL,
  inc.date = TRUE,
  as = c("data.frame", "list")
)
```

Arguments

startsWith	one letter that the package name starts with eg. a, e, f
after	packages archived after a specific date eg. 2011-05-10
inc.date	should archive date be included in the result
as	return result as data frame or as list

Value

a data frame or list containing listing of all archived R packages

Use case

This function allows the retrieval of various R packages archived by CRAN along with the respective latest archive date. The packages retrieved include both active and inactive R projects submitted to CRAN. When a new version of an active R package is published, the older versions of the package gets archived. In the same way, when a package is decommissioned from CRAN active projects for one reason or another, it gets archived.

Note

* The "startsWith" argument should be one letter and should be in lowercase
* If no argument is provided for "startsWith", all the packages will be retrieved
* The format of the "after" argument must be YYYY-MM-DD e.g. 2022-04-11

Examples

```
# Task 1: get archived R packages with names beginning with C or All
head(archivedPkg(startsWith = "all"), n= 10) #retrieves all packages
head(archivedPkg(startsWith = "c"), n= 10) #retrieves only packages beginning with a

# Task 2: return the packages from Task 1 without including latest archive date
res.dt2 <- archivedPkg(startsWith = "b", inc.date = FALSE)
res.dt2[1:10,]

# Task 3: return the results from Task 2 as a list
res.dt3 <- archivedPkg(startsWith = "c", inc.date = FALSE, as = "list")
res.dt3$name[1:10]

res.dt3 <- archivedPkg(startsWith = "e", as = "list")
res.dt3$name[1:10]

# Task 4: return the archived packages beginning with Y archived after 2022-08-12
# Note that startsWith should be lowercase

#without archive date
yRPkg <- archivedPkg(startsWith = "y", after= NULL)
nrow(yRPkg) #number of rows returned
head(yRPkg, n = 15) #show first 15 rows

#with archive date
yRPkg2 <- archivedPkg(startsWith = "y", after= "2022-08-12")
nrow(yRPkg2) #number of rows returned
head(yRPkg2, n = 15) #show first 15 rows, notice no archive date before 2022-08-12
```

`as.boolean` *Convert boolean values between formats*

Description

Convert Yes/No to 1/0 or to TRUE/FALSE or vice versa

Usage

```
as.boolean(ds, type = 3)
```

Arguments

ds	item to convert
type	format to convert to, choices 1, 2 or 3

Details

Output various format of booleans into a specified format. Below are the options for the type argument.

type: options are as follows -

1 - Yes/No
2 - TRUE/FALSE
3 - 1/0

Value

output adhering to the format of the type provided

Examples

```
# Task: convert "yes" or "no" to format of TRUE or FALSE
as.boolean("yes",2)
as.boolean("no",2)
as.boolean("YES",2)
as.boolean("NO",2)

# Task: convert "yes" or "no" to format of 1 or 0
as.boolean("yes",3)
as.boolean("no",3)
as.boolean("YES",3)
as.boolean("NO",3)


# Task: convert 1 to format of Yes or No
as.boolean(1,1)

# Task: convert "T" to format of Yes or No
as.boolean("T",1)
```

```
# Task: convert "f" to format of TRUE or FALSE
as.boolean("f",2)


# Task: convert 1 to format of TRUE or FALSE
as.boolean(1,2)


# Task: convert "Y" or "y" to format of Yes or No
as.boolean("Y",1) #uppercase Y
as.boolean("y",1) #lowercase y


# Task: convert TRUE/FALSE to format of 1 or 0
as.boolean(TRUE,3)
as.boolean(FALSE,3)


# Task: convert TRUE/FALSE to format of Yes or No
as.boolean(TRUE,1)
as.boolean(FALSE,1)



# In case of error in argument
# as.boolean("tr",3) #NA
# as.boolean("ye",3) #NA

# vector of mixed boolean to TRUE/FALSE or 1/0
multv <- c(TRUE,"y","n","YES","yes",FALSE,"f","F","T","t")
as.boolean(multv,1) # return vector as Yes/No
as.boolean(multv,2) # return vector as TRUE/FALSE
as.boolean(multv,3) # return vector as 1/0
```

`bionic_txt` *Generate a bionic text*

Description

This function serves as a mechanism enabling the conversion of provided text into a bionic form. Users input the text, and the function, in turn, delivers the text transformed into a bionic format.

Usage

```
bionic_txt(text)
```

Arguments

`text`	input text

Details

A bionic text refers to a transformed version of a given text achieved through a specialized function designed to incorporate elements of advanced technology, enhancing both the form and content of the original input. This function operates by infusing the text with a fusion of various elements, resulting in a synthesis that transcends traditional linguistic boundaries. The function augments the text with dynamic visual representations that adapt to the reader's preferences. The goal is to create a text that not only conveys information but also engages the audience in a more immersive and interactive manner, harnessing the capabilities of modern technology to redefine the traditional concept of textual communication. An example of a bionic text could be a news article that dynamically updates with real-time data, incorporates multimedia elements, and adjusts its presentation style based on the reader's preferences, thereby offering a more enriched and personalized reading experience.

Value

bionic text

References

This idea stems from a blog article published at https://www.r-bloggers.com/2023/10/little-useless-useful-r-functions-function-for-faster-reading-with-bionic-reading/ and the original source for bionic texts may be found at https://bionic-reading.com/

Examples

```
# simple example to show a text
# transformation to bionic text

# text to transform
text1 <- "A tool for nonparametric
estimation and inference
of a non-decreasing
monotone hazard\nratio
from a right censored survival dataset."

# transform text
genbt <- bionic_txt(text1)

# print bionic text as message or cat
message(genbt)
cat(genbt)
```

---

clean *Clear environment, clear console, set work directory and load files*

Description

Shorthand to quickly clear console, clear environment, set working directory, load files

Usage

```
clean(setwd = NULL, source = c(), load = c(), clearPkgs = FALSE)
```

Arguments

setwd	OPTIONAL. set working directory
source	OPTIONAL. source in file(s)
load	OPTIONAL. load in Rdata file(s)
clearPkgs	Clear previous loaded packages, TRUE or FALSE

Details

The purpose of this function is provide a one-line code to clear the console, clear the environment, set working directory to a specified path, source in various files into the current file, and load RData files into the current environment. The first process in the sequence of events is to clear the environment. Then the working directory is set, prior to inclusion of various files and RData. With the directory being set first, the path to the sourced in or RData files will not need to be appended to the file name. See examples.

Value

cleared environment and set directory

Examples

```
if(interactive()){
#simply clear environment, clear console and devices
quickcode::clean()

#clear combined with additional arguments
quickcode::clean(
  clearPkgs = FALSE
) #also clear all previously loaded packages if set to true

quickcode::clean(
  setwd = "/home/"
) #clear env and also set working directory


quickcode::clean(
  source = c("/home/file1.R","file2")
) #clear environment and source two files into current document


quickcode::clean(
  setwd = "/home/",
  source = c("file1","file2")
) #clear environment, set working directory and source 2 files into environment
```

```
quickcode::clean(
  setwd = "/home/",
  source="file1.R",
  load="obi.RData"
) #clear environment, set working directory, source files and load RData
}
```

`compHist` *Compare histograms of two distributions*

Description

For comparing histograms of two data distributions. Simply input the two distributions, and it generates a clear and informative histogram that illustrates the differences between the data.

Usage

```
compHist(
  x1,
  x2,
  title,
  col1 = "red",
  col2 = "yellow",
  xlab = "",
  ylab = "Frequency",
  separate = FALSE
)
```

Arguments

`x1`	NUMERIC. the first distribution
`x2`	NUMERIC. the second distribution
`title`	CHARACTER. title of the histogram plot
`col1`	CHARACTER. color fill for first distribution
`col2`	CHARACTER. color fill for second distribution
`xlab`	CHARACTER. label of the x-axis
`ylab`	CHARACTER. label of the y-axis
`separate`	LOGICAL. whether to separate the plots

Details

Users have the option to view individual histograms for each distribution before initiating the comparison, allowing for a detailed examination of each dataset's characteristics. This feature ensures a comprehensive understanding of the data and enhances the user's ability to interpret the results of the distribution comparison provided by this function.

Value

return histogram comparison using basic histogram plot

Some recommended color pairs

col1 = 'dodgerblue4' (and) col2 = 'darksalmon'
col1 = 'brown' (and) col2 = 'beige'
col1 = 'pink' (and) col2 = 'royalblue4'
col1 = 'red' (and) col2 = 'yellow'
col1 = 'limegreen' (and) col2 = 'blue'
col1 = 'darkred' (and) col2 = 'aquamarine4'
col1 = 'purple' (and) col2 = 'yellow'

Note

- Hexadecimal values can also be passed
in for col1 and col2, see the example section - For best visual results,
col1 should be a dark color and col2 should be passed as a light color.
For example, col1 = "black", col2 = "yellow"

Examples

```
# compare two normal distributions with means that differ a lot
# in this case, the overlap will not be observed
set.seed(123)
compHist(
  x1 = rnorm(1000, mean = 3),
  x2 = rnorm(1000, mean = 10),
  title = "Histogram of Distributions With Means 3 & 10",
  col1 = "yellow", col2 = "violet"
)


# compare two normal distributions with means that are close
# in this case, the overlap between the histograms will be observed
set.seed(123)
compHist(
  x1 = rnorm(1000, mean = 0),
  x2 = rnorm(1000, mean = 2),
  title = "Histogram of rnorm Distributions With Means 0 & 2",
  col1 = "lightslateblue", col2 = "salmon"
)

set.seed(123)
# separate the plots for preview
compHist(
  x1 = rnorm(1000, mean = 0),
  x2 = rnorm(1000, mean = 2),
  title = c("Plot Means 0", "Plot Means 2"),
  col1 = "#F96167", col2 = "#CCF381",
```

```
  separate = TRUE
)
```

`data_pop` *Remove last n rows or column or specified elements from a data frame like array_pop in PHP*

Description

Shorthand to remove elements from a data frame and save as the same name

Usage

```
data_pop(., n = 1, which = c("rows", "cols"), ret = FALSE)
```

Arguments

`.`	parent data
`n`	number of elements to remove
`which`	whether to remove from row or from column
`ret`	TRUE or FALSE. whether to return value instead of setting it to the parent data

Value

data with elements removed

Examples

```
data.01 <- mtcars[1:7,]

#task: remove 1 element from the end of the data and set it to the data name
data.01 #data.01 data before pop
data_pop(data.01) #does not return anything
data.01 #data.01 data updated after pop

#task: remove 3 columns from the end of the data and set it to the data name
data.01 #data.01 data before pop
data_pop(data.01, n = 3, which = "cols") #does not return anything, but updates data
data.01 #data.01 data updated after pop

#task: remove 5 elements from the end, but do not set it to the data name
data.01 #data.01 data before pop
data_pop(data.01,5, ret = TRUE) #return modified data
data.01 #data.01 data remains the same after pop
```

data_pop_filter *Remove elements from a data matching filter*

Description

Shorthand to remove elements from a data frame based on filter and save as the same name

Usage

```
data_pop_filter(., remove)
```

Arguments

.	data object
remove	expression for filter

Value

data filtered out based on the expression

Examples

```
# this function removes rows matching the filter expression
data.01 <- mtcars
data.02 <- airquality

#task: remove all mpg > 20
data.01 #data.01 data before pop
data_pop_filter(data.01,mpg > 15) #computes and resaves to variable
#note: this is different from subset(data.01,data.01$mpg > 15)
data.01 #modified data after pop based on filter

#task: remove all multiple. remove all elements where Month   == 5 or Solar.R > 50
data.02 #data.02 data before pop
data_pop_filter(data.02,Month   == 5 | Solar.R > 50) #computes and resaves to variable
data.02 #modified data after pop based on filter
```

data_push *Add data to another data like array_push in PHP*

Description

Shorthand to add data to a dataset and save as the same name

Usage

```
data_push(., add, which = c("rows", "cols"))
```

Arguments

.	first data set
add	data set to add
which	where to append the new data e.g. rows or cols

Value

the combined dataset store to a variable with the name of the first

Examples

```
# initialize p1 and p2
init(p1,p2)
p1
p2

# declare p1 and p2 as data frame
p1 <- data.frame(PK=1:10,ID2=1:10)
p2 <- data.frame(PK=11:20,ID2=21:30)

p1
p2

#add p1 to p2 by row, and resave as p1
data_push(p1,p2,"rows")
# p2 # p2 remains the same
p1 #p1 has been updated

# declare a new data frame called p3
p3 <- data.frame(Hindex=number(20),Rindex=number(20,seed=20))

# add p3 to p1 as column, and resave as p1
data_push(p1,p3,"cols")
p1 # p1 has been updated
```

---

data_shuffle *Shuffle a data frame just like shuffle in PHP*

Description

Shorthand to shuffle a data frame and save

Usage

```
data_shuffle(., which = c("rows", "cols"), seed = NULL)
```

Arguments

.	data to shuffle as data frame
which	what to shuffle, rows or columns
seed	apply seed if indicated for reproducibility

Value

shuffled data frame of items store to the data frame name

Examples

```
df1<-data.frame(ID=46:55,PK=c(rep("Treatment",5),rep("Placebo",5)))

#illustrate basic functionality
data_shuffle(df1)
df1 #shuffle and resaved to variable

data.f2<-df1
data_shuffle(data.f2)
data.f2 #first output

data.f2<-df1
data_shuffle(data.f2)
data.f2 # different output from first output top

data.f2<-df1
data_shuffle(data.f2,seed = 344L)
data.f2 #second output

data.f2<-df1
data_shuffle(data.f2,seed = 344L)
data.f2 #the same output as second output top
```

---

duplicate *Duplicate a file with global text replace*

Description

Shorthand to return a re-sample number of rows in a data frame by unique column

Usage

```
duplicate(file, new.name, pattern = NULL, replacement = NULL, open = TRUE)
```

Arguments

file	data frame to re-sample
new.name	column to uniquely re-sample
pattern	number of rows to return
replacement	unique numeric value for reproducibility
open	description

Value

data frame containing re-sampled rows from an original data frame

Examples

```
if(interactive()){
# example to duplicate a file, and replace text1 within it
# NOTE that, by default, this function will also open the file within RStudio

#create sample file
file1s <- paste0(tempfile(),".R")
writeLines("message(
'Sample items: eggs, coke, fanta, book'
)", file1s)

file2s <- paste0(tempfile(),".R")
file3s <- paste0(tempfile(),".R")

duplicate(
  file = file1s,
  new.name = file2s,
  pattern = 'text1',
  replacement = 'replacement1'
 )

# duplicate the file, with multiple replacements
# replace 'book' with 'egg' and 'coke' with 'fanta'
duplicate(
  file1s, file2s,
  pattern = c('book','coke'),
  replacement = c('egg','fanta')
)



# duplicate the file with no replacement
duplicate(file1s,file3s) # this simply performs file.copy, and opens the new file



# duplicate the file but do not open for editing
```

```
duplicate(file1s,file3s, open = FALSE) # this does not open file after duplication
}
```

---

genRandImg *Download random images from the web*

Description

Generate n number of high-definition images by category from the web

Usage

```
genRandImg(
  fp,
  cat = imageCategories,
  n = 1,
  w.px = 500,
  h.px = 500,
  ext = "jpg"
)
```

Arguments

fp	CHARACTER. storage directory
cat	CHARACTER. category of image to download
n	NUMERIC. number of images to download, maximum n is 99
w.px	NUMERIC. width in pixels
h.px	NUMERIC. height in pixels
ext	CHARACTER. file extension eg jpg, png

Value

downloaded image from a select image category

Sources & References

The random images are downloaded from www.unsplash.com

Category Choices

Categories for 'cat' argument include "3D", "animals", "architecture", "backgrounds", "beauty", "experimental", "fashion", "film", "food", "interior", "nature", "people", "renders", "school", "sports", "travel", "unsplash", "wallpapers".

Image categories can be captured in a separate vector as a cross-reference made available to the

cat argument.
For example:
imgcat= c("3D", "animals", "architecture", "backgrounds", "beauty", "experimental", "fashion", "film", "food", "interior", "nature", "people", "renders", "school", "sports", "travel", "unsplash", "wallpapers")

genRandImg(fp, cat = imgcat[9], n = 5)

Use case

This functionality is great for developers trying to obtain one or more images for use in displays/analysis or simply to build robust web applications.

Examples

```
# download 2 image from the nature category
genRandImg(fp = tempdir(),cat = "nature", n = 2)

# download 4 random images with width = 600px and hight 100px
genRandImg(
  fp = tempdir(),
  cat = "fashion",
  w.px = 600,
  h.px = 100)

# download 10 random images with extension png
genRandImg(fp = tempdir(),cat = "food", n = 10, ext = "png")


# download 200 random images from category of school
# Note that maximum download is 99, so the function will only download 99
genRandImg(fp = tempdir(),cat = "school", n = 200)
```

---

geo.cv *Calculate geometric coefficient of variation and round*

Description

Calculate the coefficient of variation and round

Usage

```
geo.cv(num, na.rm = TRUE, neg.rm = TRUE, pct = TRUE, round = 2)
```

Arguments

num	vector of numbers
na.rm	remove NAs from the vector
neg.rm	remove negative values from the vector
pct	TRUE or FALSE. should result be in percent
round	round result to decimal place

Value

the geometric cv of a set of numbers

Examples

```
#simulate numbers using a fixed seed
num1 <- number(n = 1115,max.digits = 4, seed = 10)

#get geometric CV, represent as percent and round to 2 decimal places
geo.cv(num1,round = 2) # result: 60.61%

#or round to 3 decimal places
geo.cv(num1,round = 3) # result: 60.609%

#by default, the above examples return a CV%
#if you do not want the result as percentage, specify "pct"
geo.cv(num1,pct = FALSE) # result: 0.61
```

---

geo.mean *Calculate geometric mean and round*

Description

Calculate the geometric mean

Usage

```
geo.mean(num, na.rm = TRUE, neg.rm = TRUE, round = 2)
```

Arguments

num	vector of numbers
na.rm	remove NAs from the vector
neg.rm	remove negative values from the vector
round	round result to decimal place

Value

the geometric mean of a set of numbers

Examples

```
num1 <- sample(300:3000,10)

#get the geometric mean, excluding all negatives and round to 2
geo.mean(num1)

#or
geo.mean(num1)


#get geometric mean, but round the final value to 5 decimal places
geo.mean(num1, round = 5)
```

---

geo.sd *Calculate geometric standard deviation and round*

Description

Calculate the geometric standard deviation

Usage

```
geo.sd(num, na.rm = TRUE, neg.rm = TRUE, round = 2)
```

Arguments

num	vector of numbers
na.rm	remove NAs from the vector
neg.rm	remove negative values from the vector
round	round result to decimal place

Value

the geometric standard deviation of a set of numbers

Examples

```
num1 <- sample(330:400,20)

#get geometric SD remove negative values and round to 2 decimal places
geo.sd(num1)

#get geometric SD, DON'T remove negative values and round to 2 decimal places
```

```
geo.sd(num1,na.rm=FALSE)

#get geometric SD, remove negative values and round to 3 decimal places
geo.sd(num1,round = 3)
```

`header.rmd` *Snippet function to add header to a current Rmd opened file*

Description

Shorthand to add Rmd header

Usage

```
header.rmd()
```

Value

Inserts header content for Rmd file

Examples

```
if(interactive())
header.rmd()
```

`in.range` *If number falls within a range of values and get closest values*

Description

With a defined range of values, the function systematically examines each provided number to determine if it falls within the specified range. It may also provide the values with the range that are closest to a desired number.

Usage

```
in.range(
  value,
  range.min,
  range.max,
  range.vec = NULL,
  closest = FALSE,
  rm.na = FALSE
)
```

Arguments

value	NUMERIC. the vector of numbers to check
range.min	NUMERIC. OPTIONAL. the minimum value of the range
range.max	NUMERIC. OPTIONAL. the maximum value of the range
range.vec	NUMERIC. OPTIONAL. a vector of numbers to use for the range
closest	BOOLEAN. OPTIONAL. return closest value
rm.na	BOOLEAN. OPTIONAL. remove NA values from input

Details

The described function serves the purpose of checking whether a given number or set of numbers falls within a specified range. It operates by taking a range of values as input and then systematically evaluates each provided number to determine if it lies within the defined range. This function proves particularly useful for scenarios where there is a need to assess numeric values against predefined boundaries, ensuring they meet specific criteria or constraints. In the same manner, this function allows the user to also retrieve values within the range that are closest to each provided number.

Value

boolean to indicate if the value or set of values are within the range

Note

The argument range.vec is utilized when users opt not to employ the range.min or range.max arguments. If range.vec is specified, range.min and range.max are disregarded. It's important to note that the use of range.vec is optional.

Examples

```
# Task 1: Check if a number is within specified range
in.range(5, range.min = 3, range.max = 10) # TRUE
in.range(25, range.min = 12, range.max = 20) # FALSE

# Task 2: Check if a set of values are within a specified range
in.range(1:5, range.min = 2, range.max = 7) #
in.range(50:60, range.min = 16, range.max = 27) #


# Task 3: Check if a number is within the range of a set of numbers
in.range(5, range.vec = 1:10) # TRUE
in.range(345, range.vec = c(1001,1002,1003,1004,1005,
1006,1007,1008,1009,1010,1011,1012,1013,1014)) # FALSE

# Task 4: Check if a set of values are within the range of a set of numbers
in.range(1:5, range.vec = 4:19) #
in.range(50:60, range.vec = c(55,33,22,56,75,213,120)) #

# Task 5: remove NAs prior to processing
in.range(c(1,3,NA,3,4,NA,8), range.min = 4, range.max = 6, rm.na = FALSE) # do not remove NA
```

```
in.range(c(1,3,NA,3,4,NA,8), range.min = 4, range.max = 6, rm.na = TRUE) # remove NA
#in.range(c(NA), range.min = 4, range.max = 6, rm.na = TRUE) #This will return error

# Task 6: return the closest number to the value
in.range(5:23, range.vec = 7:19, closest = TRUE)
in.range(-5:10, range.vec = -2:19, closest = TRUE)
in.range(c(1:5,NA,6:9), range.vec = 4:19, closest = TRUE)
in.range(c(1:5,NA,6:9), range.vec = 4:19, closest = TRUE, rm.na = TRUE)
```

`inc` *Increment vector by value*

Description

Increment the content of a vector and re-save as the vector

Usage

```
inc(., add = 1L)
```

Arguments

.	vector of number(s)
add	number to add

Details

This function is very useful when writing complex codes involving loops. Apart from the for loop, this can be useful to quickly increment a variable located outside the loop by simply incrementing the variable by 1 or other numbers. Check in the example section for a specific use. Nonetheless, one may also choose to use this function in any other instance, as it's simple purpose is to increase the value of a variable by a number and then re-save the new value to that variable.

Value

a vector incremented by a number

Examples

```
num1 <- sample(330:400,10)
num1#before increment

# increment num1 by 1
inc(num1)
num1 #after increment

# increment num1 by 5
num1 #before increment
```

```
inc(num1, add= 10)
num1 #after increment

#when used in loops

#add and compare directly
rnum = 10
inc(rnum) == 11 #returns TRUE
rnum #the variable was also updated

# use in a for loop
ynum = 1
for( i in c("scientist","dancer","handyman","pharmacist")){
message("This is the item number ")
message(ynum)
message(". For this item, I am a ")
message(i)

#decrement easily at each turn
plus(ynum)
}


#use in a repeat loop
xnum = 1
repeat{ #repeat until xnum is 15
message(xnum)
if(inc(xnum) == 15) break
}
```

`init` *Initialize new variables and objects*

Description

Shorthand to initialize one or more objects

Usage

```
init(..., value = NULL)
```

Arguments

`...`	variable names to initialize
`value`	value to initialize them to

Value

initialized objects set to the value specified

Examples

```
init(t,u,v)
message(t) # t = NULL
message(u) # u = NULL
message(v) # v = NULL
init(j,k,m,value = 7)
message(j) # j = 7
message(k) # k = 7
message(m) # m = 7
```

`insertInText` *Shiny app function to insert string to current file in RStudio*

Description

Shorthand to insert content to opened file

Usage

```
insertInText(string)
```

Arguments

`string`	what to insert

Value

Inserts into current position on opened file

Examples

```
if(interactive()){
insertInText('hello rpkg.net')
insertInText('hello world')
}
```

`is.image` *Is file name extension(s) an image*

Description

Check if one or multiple file name entry is an image

Usage

```
is.image(x)
```

Arguments

`x`	vector entry

Details

This current function tests if the extension of the file name provided belongs to any of the image extensions listed below
AI - Adobe Illustrator
BMP - Bitmap Image
CDR - Corel Draw Picture
CGM - Computer Graphics Metafile
CR2 - Canon Raw Version 2
CRW - Canon Raw
CUR - Cursor Image
DNG - Digital Negative
EPS - Encapsulated PostScript
FPX - FlashPix
GIF - Graphics Interchange Format
HEIC - High-Efficiency Image File Format
HEIF - High-Efficiency Image File Format
ICO - Icon Image
IMG - GEM Raster Graphics
JFIF - JPEG File Interchange Format
JPEG - Joint Photographic Experts Group
JPG - Joint Photographic Experts Group
MAC - MacPaint Image
NEF - Nikon Electronic Format
ORF - Olympus Raw Format
PCD - Photo CD
PCX - Paintbrush Bitmap Image
PNG - Portable Network Graphics
PSD - Adobe Photoshop Document
SR2 - Sony Raw Version 2
SVG - Scalable Vector Graphics
TIF - Tagged Image File

TIFF - Tagged Image File Format
WebP - Web Picture Format
WMF - Windows Metafile
WPG - WordPerfect Graphics

Value

a boolean value to indicate if entry is an image

Examples

```
img.1 <- "fjk.jpg"
is.image(img.1)

img.0 <- "fjk.bbVG"
is.image(img.0)

img.2 <- "fjk.bmp"
is.image(img.2)

img.3 <- "fjk.SVG"
is.image(img.3)

# a vector of file names
v <- c("logo.png", "business process.pdf",
"front_cover.jpg", "intro.docx",
"financial_future.doc", "2022 buybacks.xlsx")

is.image(v)


# when the file name has no extension
# the function returns NA
v2 <- c("img2.jpg","northbound.xlsx","landimg",NA)
is.image(v2)
```

---

libraryAll *Load specific R libraries and clear environment*

Description

Load specific packages, print a list of the loaded packages along with versions. Only include libraries, don't install if library doesn't exist

Usage

```
libraryAll(..., lib.loc = NULL, quietly = FALSE, clear = TRUE)
```

Arguments

...	multiple library names
lib.loc	OPTIONAL. library store location
quietly	OPTIONAL. attach library quietly
clear	OPTIONAL. clear environment after attach

Value

loaded libraries and clear environment

Examples

```
libraryAll(base) #one package

libraryAll(
  base,
  tools,
  stats
) #multiple packages

libraryAll("grDevices") #with quotes

libraryAll(
  stats,
  utils,
  quietly = TRUE
) #load quietly

libraryAll(
  base,
clear = FALSE) #do not clear console after load
```

---

list_push *Add elements to a list like array_push in PHP*

Description

Shorthand to add elements to a vector and save as the same name

Usage

```
list_push(., add)
```

Arguments

.	first list
add	list to add

Value

vector combining fist and second vector, but have name set to the first

Examples

```
num1 <- list(sample(330:400,10))
num2 <-list("rpkg.net")
list_push(num1, add= num2)
```

`list_shuffle` *Shuffle a list object just like shuffle in PHP*

Description

Shorthand to shuffle a list and save

Usage

```
list_shuffle(., seed = NULL)
```

Arguments

.	list to shuffle
seed	apply seed if indicated for reproducibility

Value

shuffled list of items store to the list name

Examples

```
list001 <- list("a" = 1:5,
           "b" = letters[1:5],
           c = LETTERS[1:10],
           "2" = number(5,5),
           "e" = randString(5,5))
list001 #show initial list

#illustrate basic functionality
list_shuffle(list001)
list001 #shuffle and resaved to variable

list.f2<-list001
list_shuffle(list.f2)
list.f2 #first output

list.f2<-list001
list_shuffle(list.f2)
list.f2 # different output from first output top
```

```
list.f2<-list001
list_shuffle(list.f2,seed = 344L)
list.f2 #second output

list.f2<-list001
list_shuffle(list.f2,seed = 344L)
list.f2 #the same output as second output top
```

`minus` *Decrease vector by value*

Description

decrease the content of a vector and re-save as the vector

Usage

```
minus(., minus = 1L)
```

Arguments

`.`	vector of number(s)
`minus`	number to minus

Details

Similar to the inc and plus functions, the minus function is very useful when writing complex codes involving loops. Apart from the for loop, minus can be useful to quickly decrease the value of a variable located outside the loop by simply decreement the variable by 1 or other numbers. Check in the example section for a specific use. Given the scope, one may also choose to use this function in any other instances, as it's simple purpose is to decrease the value of a variable by a number and then re-save the new value to that variable.

Value

a vector decreased by a number

Examples

```
num1 <- sample(5:150,10)
num1

# decrease num1 by 1
num1 #before decrease
minus(num1)
num1 #after decrease
```

```
# decrease num1 by 5
num1 #before decrease
minus(num1, minus = 5)
num1 #after decrease



#when used in loops

#add and compare directly
rnum = 23
minus(rnum) == 220 #returns FALSE
rnum #the variable was also updated


# use in a for loop
ynum = 100

for( i in c("teacher","student","lawyer","pharmacist")){
message("This is the item number ")
message(ynum)
message(". For this item, I am a ")
message(i)

#decrement easily at each turn
minus(ynum,3)
}

#use in a repeat loop
xnum = 100
repeat{ #repeat until xnum is 85
message(xnum)
if(minus(xnum) == 85) break
}
```

`mix.color` *Mix or Blend two or more colors*

Description

Combine colors to generate a new color

Usage

```
mix.color(color, type = 2, alpha = 1)
```

Arguments

`color`	CHARACTER. color vector e.g see example
`type`	NUMERIC. return type of the output
`alpha`	NUMERIC. alpha or opacity of the resulting color

Value

hex for the combined color

Examples

```
# color vector
colvec <- c("red", "blue", "violet", "green", "#ff0066")

# just one color
mix.color(colvec[1], type = 1, alpha = 1)

# add two colors
mix.color(colvec[1:2], type = 1, alpha = 1)

# add three colors
mix.color(colvec[1:3], type = 1, alpha = 1)


# return type = 2

# just one color
mix.color(colvec[1], type = 2, alpha = 1)

# add two colors
mix.color(colvec[1:2], type = 2, alpha = 1)

# add three colors
mix.color(colvec[1:3], type = 2, alpha = 1)


# opacity or alpha  0.5

# just one color
mix.color(colvec[1], type = 1, alpha = 0.5)

# add two colors
mix.color(colvec[1:2], type = 1, alpha = 0.5)

# add three colors
mix.color(colvec[1:3], type = 1, alpha = 0.5)

# add all colors
mix.color(colvec, type = 1, alpha = 0.5)
```

`mix.cols.btw` *Mix or Blend colors between two or more colors*

Description

Mix or blend multiple colors between two colors

Usage

```
mix.cols.btw(colors, max = 20, alpha = 1, preview = FALSE)
```

Arguments

colors	the vector of two colors
max	maximum number of colors to blend between
alpha	alpha for the new color blends
preview	LOGICAL. preview all color generated

Value

color hex for all generated colors

Examples

```
# simply mix/blend two colors
mix.cols.btw(c("red","brown"))

# simply mix/blend two colors, maximum number of colors at the end
mix.cols.btw(c("red","brown"), max = 8)

# simply mix/blend two colors with alpha=0.2 (opacity=0.2)
mix.cols.btw(c("yellow","green"),alpha = 0.2)

# also preview after mixing the two colors
mix.cols.btw(c("red","green"), preview = TRUE)
mix.cols.btw(c("blue","violet"),alpha = 0.2, preview = TRUE)

mix.cols.btw(c("red","purple","yellow","gray"), preview = TRUE)

mix.cols.btw(c("red","purple","yellow","gray"),alpha = 0.2, preview = TRUE)
```

---

not.data *Not a data*

Description

Check if entry is not a data object

Usage

```
not.data(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is a data table

Examples

```
test.dt <- data.frame(ID=1:200,Type="RPKG.net")
test.notenv <- list(t=1)
not.data(test.dt) # FALSE
not.data(test.notenv) # TRUE
if(not.data(test.dt)) message("yes") # NULL
```

---

not.duplicated *Not duplicated elements*

Description

Checks which elements of a vector or data frame are NOT duplicates of elements with smaller subscripts

Usage

```
not.duplicated(x, incomparables = FALSE, ...)
```

Arguments

x	a vector or a data frame or an array or NULL.
incomparables	a vector of values that cannot be compared. FALSE is a special value, meaning that all values can be compared, and may be the only value accepted for methods other than the default. It will be coerced internally to the same type as x
...	arguments for particular methods.

Value

elements of a vector or data frame that are NOT duplicates

Examples

```
set.seed(08082023)
dtf <- sample(1:10,15, replace = TRUE)
dtf # 3  9 10  3  8  9  6 10  5  1  2  2  2  9  8
dtf[ dtf > 4 & not.duplicated(dtf) ] # 9 10  8  6  5
```

not.empty *Not empty*

Description

Check if entry is not empty

Usage

```
not.empty(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is empty

Examples

```
not.empty("empty") # TRUE
not.empty('') # FALSE
not.empty(y<-NULL) # FALSE
if(not.empty('')) message("yes") # NULL
```

not.environment *Not an environment*

Description

Check if entry is not an environment object

Usage

```
not.environment(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is an environment

Examples

```
test.env <- new.env()
test.notenv <- list(t=1)
not.environment(test.env) # FALSE
not.environment(test.notenv) # TRUE
if(not.environment(test.notenv)) message("yes") # yes
```

`not.image` *File name extension(s) is Not an image*

Description

Check if one or multiple file name entry is not an image

Usage

```
not.image(x)
```

Arguments

x	vector entry

Details

This current function tests if the extension of the file name provided does NOT belongs to any of the image extensions listed below
AI - Adobe Illustrator
BMP - Bitmap Image
CDR - Corel Draw Picture
CGM - Computer Graphics Metafile
CR2 - Canon Raw Version 2
CRW - Canon Raw
CUR - Cursor Image
DNG - Digital Negative
EPS - Encapsulated PostScript
FPX - FlashPix
GIF - Graphics Interchange Format
HEIC - High-Efficiency Image File Format
HEIF - High-Efficiency Image File Format
ICO - Icon Image
IMG - GEM Raster Graphics
JFIF - JPEG File Interchange Format
JPEG - Joint Photographic Experts Group
JPG - Joint Photographic Experts Group
MAC - MacPaint Image
NEF - Nikon Electronic Format

ORF - Olympus Raw Format
PCD - Photo CD
PCX - Paintbrush Bitmap Image
PNG - Portable Network Graphics
PSD - Adobe Photoshop Document
SR2 - Sony Raw Version 2
SVG - Scalable Vector Graphics
TIF - Tagged Image File
TIFF - Tagged Image File Format
WebP - Web Picture Format
WMF - Windows Metafile
WPG - WordPerfect Graphics

Value

a boolean value to indicate if entry is not an image

Examples

```
img.1 <- "fjk.jpg"
not.image(img.1)

img.2 <- "fjk.bmp"
not.image(img.2)

img.3 <- "fjk.SVG"
not.image(img.3)

# a vector of file names
v <- c("logo.png", "business process.pdf",
"front_cover.jpg", "intro.docx",
"financial_future.doc", "2022 buybacks.xlsx")
not.image(v)

# when the file name has no extension
# the function returns NA
v2 <- c("img2.jpg",NA,"northbound.xlsx","landimg")
not.image(v2)
```

---

not.integer *Not an integer*

Description

Check if entry is not an integer

Usage

```
not.integer(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is an integer

Examples

```
not.integer(23.43) # TRUE
not.integer(45L) # FALSE
if(not.integer(4L)) message("yes") # NULL
```

---

not.logical *Not logical*

Description

Check if entry is a logical object

Usage

```
not.logical(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is logical

Examples

```
test.env <- TRUE
test.notenv <- 0
not.logical(test.env) # FALSE
not.logical(test.notenv) # TRUE
if(not.logical(test.notenv)) message("yes") # yes
```

not.na *Not NA*

Description

Check if entry is not NA

Usage

```
not.na(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is NA

Examples

```
not.na(NA) # FALSE
not.na(NULL) # logical(0)
if(not.na(45)) message("something") # TRUE
```

not.null *Not NULL*

Description

Check if entry is not NULL

Usage

```
not.null(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is NULL

Examples

```
not.null("") # TRUE
not.null(NULL) # FALSE
if(not.null(45)) message("something") # yes
```

`not.numeric` *Not numeric*

Description

Check if entry is not numeric

Usage

```
not.numeric(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is numeric

Examples

```
not.numeric("45") # TRUE
not.numeric(45) # FALSE
if(not.numeric(45)) message("yes") # yes
```

`not.vector` *Not a vector*

Description

Check if entry is not vector

Usage

```
not.vector(x)
```

Arguments

x	vector entry

Value

a boolean value to indicate if entry is vector

Examples

```
vect1 = list(r=1,t=3:10)
vect2 = LETTERS
not.vector(vect1) # FALSE
not.vector(vect2) # FALSE
if(not.vector(vect1)) message("yes") # NULL
```

---

number *Generate a random number*

Description

Shorthand code to generate a random number

Usage

```
number(n, max.digits = 10, seed = NULL)
```

Arguments

n	how many numbers to generate
max.digits	maximum number of digits in each number
seed	set seed for sampling to maintain reproducibility

Value

random numbers between 1 and 1 billion

Examples

```
number(1)
number(10)
paste0(number(2),LETTERS)

#set maximum number of digits
number(1,max.digits = 5)
number(10,max.digits = 4)

#set seed for reproducibility
#without seed
number(6) #result 1
number(6) #result 2, different from result 1
#with seed
number(6,seed=1)#result 3
number(6,seed=1)#result 4, same as result 3
```

plus *Increment vector by value*

Description

Increment the content of a vector and re-save as the vector

Usage

```
plus(., add = 1L)
```

Arguments

.	vector of number(s)
add	number to add

Details

This function is very useful when writing complex codes involving loops. Apart from the for loop, this can be useful to quickly increment a variable located outside the loop by simply incrementing the variable by 1 or other numbers. Check in the example section for a specific use. Nonetheless, one may also choose to use this function in any other instance, as it's simple purpose is to increase the value of a variable by a number and then re-save the new value to that variable.

Value

a vector incremented by a number

Examples

```
num1 <- sample(330:400,10)
num1#before increment

# increment num1 by 1
inc(num1)
num1 #after increment

# increment num1 by 5
num1 #before increment
inc(num1, add= 10)
num1 #after increment

#when used in loops

#add and compare directly
rnum = 10
inc(rnum) == 11 #returns TRUE
rnum #the variable was also updated
```

```
# use in a for loop
ynum = 1
for( i in c("scientist","dancer","handyman","pharmacist")){
message("This is the item number ")
message(ynum)
message(". For this item, I am a ")
message(i)

#decrement easily at each turn
plus(ynum)
}


#use in a repeat loop
xnum = 1
repeat{ #repeat until xnum is 15
message(xnum)
if(inc(xnum) == 15) break
}
```

---

randString *Generate a random string*

Description

Create a random string of specified length

Usage

```
randString(n, length)
```

Arguments

n	number of strings to create
length	length of string to create

Value

one more random string of specific length

Examples

```
# Task 1: create 1 random string string of length 5
randString(n = 1, length = 5)

# Task 2: create 5 random string string of length 10
randString(n = 5, length = 10)
```

```
# Task 3: create 4 random string string of length 16
randString(n = 4, length = 16)
```

`rcolorconst` *R Color Constant*

Description

This function provides information that describes the color constants that exist in R

Usage

```
rcolorconst(title = "R Color Constants")
```

Arguments

`title`	title of the output

Details

In addition to the color palette in R that can be represented as either color literals or hexadecimal values, numeric values can also be used to add colorization to a plot. Numeric values ranging from 1 to 8 provide 8 basic colors that can be deployed. The rcolorconst function returns both a Named Vector and a color palette plot that connects these numeric values with their corresponding color.

Value

returns color constant

Examples

```
# Without title
ex1 <- rcolorconst()

# With title
ex2 <- rcolorconst("My new color constant")

# More detailed example
set.seed(200)
x = data.frame(
  meas = rnorm(100),
  grp = sample(1:8, size = 100,
  replace = TRUE))
plot(x, pch = 16, col = x$grp)
colnums = rcolorconst()
```

rDecomPkg *Check whether an R package has been decommissioned in CRAN*

Description

Designed to assist users in checking the decommission status of an R package on CRAN. In the context of R language, CRAN stands for the Comprehensive R Archive Network.

Usage

```
rDecomPkg(package)
```

Arguments

package	package name to query

Details

CRAN is a network of servers around the world that store R packages and their documentation, providing a centralized repository for the R community. With the current function, users can quickly and easily determine whether a specific R package has been decommissioned on CRAN, ensuring they stay informed about the availability and support status of the packages they rely on for their R programming projects. This tool simplifies the process of package management, helping users maintain up-to-date and reliable dependencies in their R code.

Value

the decommissioned status of a particular package based on the available packages using the utils package

Examples

```
# check if cattonum package is decomissioned
# the current package is expected to be decommissioned
rDecomPkg("cattonum")

# check if dplyr is decommissioned
# the current package is expected NOT to be decommissioned
rDecomPkg("dplyr")
```

refresh *Clear environment, clear console, set work directory and load files*

Description

Shorthand to quickly clear console, clear environment, set working directory, load files

Usage

```
refresh(setwd = NULL, source = c(), load = c(), clearPkgs = FALSE)
```

Arguments

setwd	OPTIONAL. set working directory
source	OPTIONAL. source in file(s)
load	OPTIONAL. load in Rdata file(s)
clearPkgs	clear previously loaded packages

Details

The purpose of this function is provide a one-line code to clear the console, clear the environment, set working directory to a specified path, source in various files into the current file, and load RData files into the current environment. The first process in the sequence of events is to clear the environment. Then the working directory is set, prior to inclusion of various files and RData. With the directory being set first, the path to the sourced in or RData files will not need to be appended to the file name. See examples.

Value

cleared environment and set directory

Examples

```
if(interactive()){
#exactly like the clean function
#simply clear environment, clear console and devices
quickcode::refresh()

#clear combined with additional arguments
quickcode::refresh(
  clearPkgs = FALSE
) #also clear all previously loaded packages if set to TRUE

quickcode::refresh(
  setwd = "/home/"
) #clear env and also set working directory
```

```
quickcode::refresh(
  source = c("/home/file1.R","file2")
) #clear environment and source two files into current document


quickcode::refresh(
  setwd = "/home/",
  source = c("file1","file2")
) #clear environment, set working directory and source 2 files into environment


quickcode::refresh(
  setwd = "/home/",
  source="file1.R",
  load="obi.RData"
) #clear environment, set working directory, source files and load RData

}
```

`sample_by_column` *Re-sample a dataset by column and return number of entry needed*

Description

Shorthand to return a re-sample number of rows in a data frame by unique column

Usage

```
sample_by_column(.dt, col, n, seed = NULL, replace = FALSE)
```

Arguments

`.dt`	data frame to re-sample
`col`	column to uniquely re-sample
`n`	number of rows to return
`seed`	unique numeric value for reproducibility
`replace`	should sampling be with replacement

Value

data frame containing re-sampled rows from an original data frame

Examples

```
data1 <- data.frame(ID=1:10,MOT=11:20)
sample_by_column(data1,MOT,3)
sample_by_column(data1,ID,7)
```

`setOnce` *Set a variable only once*

Description

Facilitates the one-time setting of a variable in R, ensuring its immutability thereafter.

Usage

```
setOnce(., val = 1L)
```

Arguments

`.`	variable to set
`val`	the value to set for the variable

Details

With this function, users can establish the change to the initial value of a variable, and it guarantees that any subsequent attempts to modify the variable are ignored. This feature ensures that the variable remains constant and immutable once it has been set, preventing unintentional changes and promoting code stability. This function simplifies the process of managing immutable variables in R, providing a reliable mechanism for enforcing consistency in data throughout the course of a program or script.

Value

the variable set to the new variable, along with a class of once added to the output

Examples

```
# set the value of vector_x1, vector_y1, vector_z1
init(vector_x1, vector_y1, vector_z1, value = 85)

# view the initial values of the variables
vector_x1
vector_y1
vector_z1

# task 1: change the value vector_x1 and prevent further changes
vector_x1 # check value of unchanged
vector_x1 * 0.56 # check value when x 0.56

setOnce(vector_x1, val = 4500) # set vector_x1
vector_x1 # check value
vector_x1 * 0.56 # check value when x 0.56

setOnce(vector_x1, val = 13) # set vector_x1 AGAIN, should not change
vector_x1 # check value
```

```
vector_x1 * 0.56 # check value when x 0.56

# task 2: In for loop, change vector_y1 and use later
vector_y1 # check value of unchanged

for(i in 1:20){
setOnce(vector_y1,as.numeric(Sys.time()))
# now let's see the difference between vector_y1
# and the current time as it changes
message("current vector_y1: ",vector_y1,"; subtraction res: ",as.numeric(Sys.time()) - vector_y1)
}

# task 3: In for lapply, change vector_z1 and use later
vector_z1 # check value of unchanged

invisible(
lapply(1:20, function(i){
setOnce(vector_z1,as.numeric(Sys.time()))
# now let's see the difference between vector_z1
# and the current time as it changes
message("current vector_z1: ",vector_z1,"; subtraction res: ",as.numeric(Sys.time()) - vector_z1)
})
)

# result of all the tasks
vector_x1
vector_y1
vector_z1
```

`vector_pop` *Remove last n elements or specified elements from a vector like array_pop in PHP*

Description

Shorthand to remove elements from a vector and save as the same name

Usage

```
vector_pop(., n = 1, el = NULL, ret = FALSE)
```

Arguments

`.`	parent vector
`n`	number of elements to remove
`el`	vector to remove
`ret`	TRUE or FALSE. whether to return value instead of setting it to the parent vector

Value

vector with elements removed

Examples

```
num1 <- sample(330:400,10)
name1 <- "ObinnaObianomObiObianom"

#task: remove 1 element from the end of the vector and set it to the vector name
num1 #num1 vector before pop
vector_pop(num1) #does not return anything
num1 #num1 vector updated after pop

#task: remove 5 elements from the end, but do not set it to the vector name
num1 #num1 vector before pop
vector_pop(num1,5, ret = TRUE) #return modified vector
num1 #num1 vector remains the same after pop


#task: remove 6 elements from a word, set it back to vector name
name1 #name1 before pop
vector_pop(name1,6) #does not return anything
name1 #name updated after pop

#task: remove 3 elements from a word, Do not set it back to vector name
name1 #name1 before pop
vector_pop(name1,3, ret = TRUE) #returns modified name1
name1 #name1 not updated after pop

#task: remove 4 elements from the end of a vector and return both the removed content and remaining
v_f_num <- paste0(number(20),c("TI")) #simulate 20 numbers and add TI suffix
v_f_num #show simulated numbers
vector_pop(v_f_num, n = 4, ret = TRUE) #get the modified vector
vector_pop(v_f_num, n = 4, ret = "removed") #get the content removed

#task: remove specific items from vector
#note that this aspect of the functionality ignores the 'n' argument
v_f_num_2 <- paste0(number(6, seed = 33),c("AB")) #simulate 6 numbers using seed and add AB suffix
v_f_num_2 #show numbers
vector_pop(v_f_num_2, el = c("403211378AB")) #remove 1 specific entries
v_f_num_2 #show results
vector_pop(v_f_num_2, el = c("803690460AB","66592309AB")) #remove 2 specific entries
v_f_num_2 #show results
```

---

vector_push *Add elements to a vector like array_push in PHP*

Description

Shorthand to add elements to a vector and save as the same name

Usage

```
vector_push(., add, unique = FALSE, rm.na = FALSE, rm.empty = FALSE)
```

Arguments

.	first vector
add	vector to add
unique	remove duplicated entries
rm.na	remove NA values
rm.empty	remove empty values

Details

Note that two vectors are required in order to use this function. Also, note that the final result replaces the content of the first vector. This means that the original content of the 'first vector' will no longer exist after this function executes.

Value

vector combining fist and second vector, but have name set to the first

Use case

This function allows the combination of two vectors in one short line of code. It allows specification of further downstream filtering of the resulting vector such as selecting only unique items, removing NA or empty values. It simplifies a code chunk with many lines of code to concatenate and filter various vectors.

Examples

```
num1 <- number(10, seed = 45)
num2 <-"rpkg.net"

num1
num2

#Task: add num2 to num1 and re-save as num1
vector_push(num1,num2)
num1 #updated with num2
num2 #not updated


#Task: concatenate two vectors and remove duplicates
vector1 = number(4,seed = 5)
vector2 = number(8,seed = 5)
vector3 = number(12,seed = 5)

vector1 #length is 4
vector2 #length is 8
```

```
vector3 #length is 12

# with duplicated
vector_push(vector1,vector2, unique = FALSE)
vector1 #return modified vector
length(vector1) #length is 12 because nothing was removed
#duplicates in vector1 is 886905927 100040083 293768998  54080431

# without duplicated
vector_push(vector2,vector3, unique = TRUE)
vector2 #return modified vector
length(vector2) #length is 12 instead of 20
#Total of 8 duplicated numbers were removed


#Task: concatenate two vector and remove NA values
vector1 = number(5)
vector2 = c(4,NA,5,NA)
vector3 = number(5)

# with NA
vector_push(vector1,vector2, rm.na = FALSE)
vector1 #return modified vector

# without NA
vector_push(vector3,vector2, rm.na = TRUE)
vector3 #return modified vector


#Task: concatenate two vector and remove empty values
vector1 = number(5)
vector2 = c(4,'',5,'',NULL,' ')
vector3 = number(5)

# with empty
vector_push(vector1,vector2, rm.empty = FALSE)
vector1 #return modified vector

# without empty
vector_push(vector3,vector2, rm.empty = TRUE)
vector3 #return modified vector
```

`vector_shuffle` *Shuffle a vector just like shuffle in PHP*

Description

Shorthand to shuffle a vector and save

Usage

```
vector_shuffle(., replace = FALSE, prob = NULL, seed = NULL)
```

Arguments

.	vector to shuffle
replace	replace selected value
prob	probability of occurrence
seed	apply seed if indicated for reproducibility

Value

shuffled vector of items store to the vector name

Examples

```
v1<-c(3,45,23,3,2,4,1)


#demonstrate vector_shuffle
vector_shuffle(v1)
v1 # show outputs

#demonstrate reproducibility in shuffle with seed
v0<-v1
vector_shuffle(v0)
v0 #first output

v0<-v1
vector_shuffle(v0)
v0 # different output from first output top

v0<-v1
vector_shuffle(v0,seed = 232L)
v0 #second output

v0<-v1
vector_shuffle(v0,seed = 232L)
v0 #the same output as second output top
```

---

yesNoBool *Convert Yes/No to Binary or Logical*

Description

Seamlessly convert a yes or no to either a binary or logical output

Usage

```
yesNoBool(
  table,
  fldname,
  out = c("change", "append", "vector"),
  type = c("bin", "log")
)
```

Arguments

table	data frame
fldname	field name in the data frame
out	output form, choices - change, append, vector
type	output type, choices - bin, log

Details

type - "bin" for binary, and "log" for logical

Value

converted Yes/No entries into 1/0 or TRUE/FALSE

Examples

```
# Declare data for example
usedata <- data.frame(ID = 1:32)
usedata #view the dataset

usedata$yess = rep(c("yes","n","no","YES","No","NO","yES","Y"),4) #create a new column
usedata #view the modified dataset

# Set all yess field as standardize boolean
# Task: convert the "yess" column content to 1/0 or TRUE/FALSE
# Notice that you have add the column name with or without quotes
yesNoBool(usedata,yess, type="bin") #set all as binary 1/0
yesNoBool(usedata,"yess", type="log") #set all as logical TRUE/FALSE


# Task: By default, the 'out' argument is set to "change"
# means that the original data field will be
# replaced with the results as above

# In this example, set the out variable to
# append data frame with a new column name containing the result

yesNoBool(usedata,yess,"append")
#or yesNoBool(usedata,"yess","append")
```

```
# In this example, return as vector
yesNoBool(usedata,yess,"vector")
#or yesNoBool(usedata,"yess","vector")

# Task: Return result as logical
yesNoBool(usedata,"yess",type = "log")
```

`%nin%` *Not in vector or array*

Description

Check if entry is in vector

Usage

```
x %nin% table
```

Arguments

`x`	vector entry
`table`	table of items to check

Value

a boolean value to indicate if entry is present

Examples

```
5 %nin% c(1:10) #FALSE
5 %nin% c(11:20) #TRUE

x = "a"
if(x %nin% letters) x

# let's say we are trying to exclude numbers from a vector
vector_num1 <- number(9, max.digits = 5, seed = 1) #simulate 9 numbers
vector_num1 #values
vector_num1[vector_num1 %nin% c(83615,85229)]#return values not 83615 or 85229
```

Index