Package: proteinAnnotation (via r-universe)

August 31, 2024

Title ProteinAnnotation

Imports magrittr, dplyr, ggforce, ggplot2, glue, purrr, scales, stringr, tidyr, rlang, DescTools, gt, r3dmol

Version 0.5.1

Author Ben Bruyneel <benbruyneel@gmail.com>

Maintainer Ben Bruyneel <benbruyneel@gmail.com>

Description Display protein information.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Repository https://benbruyneel.r-universe.dev

RemoteUrl https://github.com/BenBruyneel/proteinAnnotation

RemoteRef HEAD

RemoteSha c46d0da3c310f948d9d2721f02a1085aefc6a158

Contents

`addDataToProtein` *addDataToProtein*

Description

Takes data.frame with position data and adds it as an extra column to a protein data data.frame Positions not covered in the added data.frame are set to the argument NAValue (default is integer 0)

Usage

```
addDataToProtein(
  proteinDF,
  dataframe,
  dataColumn,
  dataName,
  positionColumn = "position",
  NAValue = 0L,
```

```
  strictPosition = TRUE
)
```

Arguments

proteinDF	the protein data.frame to add the info to. Main requirements are that there is a column named position, that there is no column named 'newData____' (internal use) and no column with the same name as defined by the 'dataName' argument
dataframe	data.frame with the data to be added. Should have two columns: one with the position info and another with the data relating to these positions
dataColumn	name of the column in dataframe with the data to be added
dataName	name of the new data column in the protein data data.frame
positionColumn	name of the position column in dataframe which has the position info (default is 'position')
NAValue	value for the position in the protein data data.frame which are missing in the (new) dataframe data
strictPosition	logical vector. Default is TRUE: if the positions in the (new) dataframe are outside the range of the protein data, then an error will be generated. If FALSE, then only a warning is generated

Value

data.frame

Examples

```
bsaProtein <- createProteinData(sequence = standardProtein("bsa"), nterm = 1,
 start = 25)
toAdd <- data.frame(position = c(25:30, 100:110), data = c(rep(1,10),
 rep(3, 3),rep(2,4)))
bsaProtein <- addDataToProtein(proteinDF = bsaProtein, dataframe = toAdd,
 dataColumn = "data", dataName = "found")
bsaProtein$position[bsaProtein$sequence == "K" | bsaProtein$sequence == "R"]
trypticSites <- data.frame(position = bsaProtein$position[
 bsaProtein$sequence == "K" | bsaProtein$sequence == "R"],
 data = "tryptic")
bsaProtein <- addDataToProtein(proteinDF = bsaProtein, dataframe = trypticSites,
 dataColumn = "data", dataName = "trypticSite", NAValue = "")
bsaProtein[95:110,]
```

---

addStyleElement *addStyleElement*

Description

takes a 3d object of the class 'r3dmol' as created via the 'r3dmol' library amd adds the colors as defined in the protein data. The function m_add_style is used for this.

Usage

```
addStyleElement(
  object3D,
  proteinDF,
  positionColumn = "position",
  colorColumn,
  colorsReduce = T,
  style = r3dmol::m_style_cartoon,
  ...
)
```

Arguments

object3D	a r3dmol object of a protein information file (pdf, cif)
proteinDF	protein data data.frame with at least two columns: position & color
positionColumn	name of the column containing the positions to be colored (default: 'position')
colorColumn	name of the column containing the colors to be used for the positions
colorsReduce	this allows for a smaller number of style objects to be added to the object3D
style	function that defines the style to be 'added' to the object3D object (default: m_style_cartoon)
...	for adding sdditional argunebts to the style-function. For 'm_style_cartoon', e.g. arrows = FALSE can be used.

Value

a r3dmol object

Examples

```
# create protein data
ovaProtein <- proteinCoverage(sequence = standardProtein(),
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", shiftPosition = 1)
# mapping Abundance information
ovaProtein <- mapPeptidesToProtein(proteinDF = ovaProtein, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins",
 shiftPosition = 1L, variable = "Abundances_1",
 dataName = "Abundance", combineFunction = sum)
# translate the colors
logColors <- translateToGradientColors(minValue = 1E3,
 maxValue = max(ovaProtein$Abundance),
 colorSteps = 1000, colors = c("white", "blue"), transformFunction = log10)
ovaProtein$AbundanceLogColors <- logColors(ovaProtein$Abundance)
library(r3dmol)
# get pdb file from alphafold
downloadedFile <- tempfile(fileext = ".pdb")
download.file("https://alphafold.ebi.ac.uk/files/AF-P01012-F1-model_v4.pdb",
 destfile = downloadedFile)
```

```
# create r3dmol object
## Not run:
p3d <- r3dmol(viewer_spec = m_viewer_spec(
  cartoonQuality = 20,
  lowerZoomLimit = 50,
  upperZoomLimit = 350
)) %>%
  m_add_model(downloadedFile) %>%
  m_set_style(style = m_style_cartoon(arrows = F)) %>%
  m_zoom_to()
p3d
# add colors
addStyleElement(p3d, proteinDF = ovaProtein, colorColumn = "AbundanceColors", arrows = FALSE)
addStyleElement(p3d, proteinDF = ovaProtein, colorColumn = "AbundanceLogColors", arrows = FALSE)

## End(Not run)
```

---

boxData *boxData*

Description

Takes a data.frame and adds a 'row' and 'col' column which together define which cell in the rows*columns-matrix a data.frame-row maps to

Usage

```
boxData(
  data,
  ncols = NA,
  adjustRows = TRUE,
  emptySequence = "",
  replaceNA = NA
)
```

Arguments

data	data.frame, should not already have a 'row' and/or 'col' column
ncols	number of required columns for the matrix
adjustRows	logical vector, default is TRUE. Defines whether the number of rows for the data.frame is allowed to be adjusted. Setting this to FALSE may lead to 'stop'-error if data doesn't fit the requested matrix.
emptySequence	if data.frame is extended, this character vector is used to sequence in the new rows. The position column is simply filled with increasing integers.
replaceNA	default is NA, if defined as something else, then ALL NA values in the data.frame will be replace with this argument. This is an admittedly crude way to replace NA values. If more finesse is needed, leave at default and handle NA values in followup functions such as 'replace_na' from the tidyr package

Value

a data.frame with extra columns (row and col) and possible extra rows

Note

If the data.frame extended: the position column and the sequence are filled with increasing integers (starting from the max position) and the 'emptySequence' argument respectively. Other columns in the extended using NA

Examples

```
boxData(createProteinData(standardProtein(), start = 2, nterm = 1), ncols = 50) |> head(20)
boxData(createProteinData(standardProtein(), start = 2, nterm = 1), ncols = 50) |> tail(20)
boxData(createProteinData(standardProtein(), start = 2, nterm = 1), ncols = 50)[40:60,]
```

`boxDimensions` *boxDimensions*

Description

Calculates/defines the dimensions of the 'matrix' to use to display data of a certain length (e.g. a protein sequence)

Usage

```
boxDimensions(
  vectorLength = 100,
  ratio = NA,
  increaseColumns = FALSE,
  ncol = NA,
  nrow = NA,
  noWarning = FALSE
)
```

Arguments

`vectorLength`	minimum number of cells for the resulting columns*rows matrix
`ratio`	the intended ratio the resulting columns*rows matrix. Increase ratio to get more columns, decrease to get more rows.
`increaseColumns`	if TRUE, then the number of columns will be increased to reach the number of matrix cells required. If FALSE then the number of rows will be increased to do this. Default is FALSE
`ncol`	defines the number of columns to use. If not also defined, the number of rows is adapted to fit all of the required number of cells for the matrix
`nrow`	defines the number of rows to use. If not also defined, the number of columns is adapted to fit all of the required number of cells for the matrix

noWarning — logical vector, if product of the number of rows and the number of columns is too small for the number of matrix cells (vector length) required then a warning will displayed (default noWarning is FALSE), unless noWarning is TRUE

Value

a named numeric vector (names: ncol, nrow)

Note

Order of importance for the ratio, ncol and nrow arguments: if the ratio is NA (default), then the ncol argument is used, if possible with the nrow argument. If ncol is NA, then the nrow argument is used.

Examples

```
boxDimensions(385, 5)
boxDimensions(385, nrow = 9)
boxDimensions(400, 10)
boxDimensions(400, ncol = 60)
```

---

containsModification *containsModification*

Description

Helper function to check whether a character vector of format modification,splitCahracter,modification,splitCahracter, (etc etc) contains a certain modification

Usage

```
containsModification(
  modifications,
  whichModification,
  strict = TRUE,
  splitCharacter = ",",
  trim = TRUE
)
```

Arguments

modifications — character vector of format etc etc

whichModification — the modification to search for (character vector)

strict — logical vector, default is TRUE: the search for the modification is strict. If FALSE then the search goes via the 'grepl' function and regular expressions can be used

splitCharacter	character vector used to split the modifications argument
trim	logical vector. Default is TRUE: the modifications will be trimmed (spaces on both sides removed). When FALSE, this is not done

Value

logical vector

Examples

```
containsModification(c('Oxidation,Phospho','Carbamidomethyl,Phospho'),
 whichModification = "Phospho")
containsModification(c('Oxidation , Phospho','Carbamidomethyl, Phospho'),
 whichModification = "Phospho")
containsModification(c('Oxidation , Phospho','Carbamidomethyl, Phospho'),
 whichModification = "Phospho", trim = TRUE)
containsModification(c('Oxidation , Phospho','Carbamidomethyl, Phospho'),
 whichModification = "Phospho", strict = FALSE)
```

---

createProteinData *createProteinData*

Description

Create a protein data.frame with basic information: position & sequence. In the resulting data.frame, every row has an amino acid (letter, sequence column) and the position of that amino acid in the protein (position column)

Usage

```
createProteinData(
  sequence,
  start = 1,
  end = nchar(sequence),
  nterm = NA,
  cterm = NA,
  emptySequence = " "
)
```

Arguments

sequence	character vector, protein sequence in the form of the usual letter codes
start	first position in the sequence to be used
end	last position in the sequence to be used
nterm	start position 'adjustment'. Adds a number of 'empty' rows to the head of the resulting data.frame to have it start at position nterm

cterm	end position 'adjustment'. Adds a number of 'empty' rows to the tail of the resulting data.frame to have it ends at position cterm
emptySequence	default " ", character vector for the sequence column to use when adding rows. Note: other columns will also be extended and usually add NA's as a value

Value

data.frame with two columns: position (integer vector) and sequence (amino acid letters: character vector)

Note

nterm stands for n-terminus, cterm stands for c-terminus

Examples

```
createProteinData(sequence = standardProtein("bsa"), start = 25) |> head(25)
createProteinData(sequence = standardProtein("bsa"), nterm = 1, start = 25) |> head(25)
createProteinData(sequence = standardProtein("bsa"), start = 25) |> tail(10)
createProteinData(sequence = standardProtein("bsa"), start = 25, nterm = 1, cterm = 610) |>
 tail(10)
```

---

dfToModPositions *dfToModPositions*

Description

Takes a single modification table row with one or more modification definitions (as produced by modPositionsToDF) and makes it into a single-modification-per-row data.frame. A row with 'x' modifications becomes 'x' number of rows. Essentially the opposite of modPositionsToDF.

Usage

```
dfToModPositions(df, splitChar = ",", returnEmptyRow = TRUE)
```

Arguments

df	a single row data.frame having the 'collapsed' columns (modifications, positions, letters and numbers) Each modification(-type) is separated by the argument 'splitChar'
splitChar	character vector specifying the separation 'character' that is placed between the modification definitions
returnEmptyRow	logical vector, default is TRUE. TRUE: if no modifications are present in the 'df' argument, a data.frame will be returned without rows. FALSE: in that case a data.frame with a single row, with in every column an empty character vector, will be returned.

Value

a data.frame with columns: modifications, positions, letters and numbers

Examples

```
modPositions(OVATable("peptide")$Modifications[32]) |> modPositionsToDF() |> dfToModPositions()
modPositions(OVATable("peptide")$Modifications[2]) |> modPositionsToDF() |> dfToModPositions()
modPositions(OVATable("peptide")$Modifications[1])
modPositions(OVATable("peptide")$Modifications[1]) |> modPositionsToDF()
modPositions(OVATable("peptide")$Modifications[1]) |> modPositionsToDF() |> dfToModPositions()
```

---

displayProtein *displayProtein*

Description

displays the protein data in a data frame via ggplot2's geom_tile

Usage

```
displayProtein(
  proteinData,
  columns = NA,
  emptySequence = "",
  replaceNA = NA,
  backGroundColor = "white",
  frontColor = "black",
  borderColor = backGroundColor,
  defaultTextColor = "black",
  textColorColumn = NA,
  replaceTextNA = "0",
  textColorLevels = NA,
  textColorLabels = NA,
  textColors = c("black"),
  textColorFunction = NA,
  showTextLegend = F,
  textLegendTitle = textColorColumn,
  textLegendDirection = "vertical",
  fillColorColumn = NA,
  replaceFillNA = NA,
  fillColorLevels = NA,
  fillColorLabels = NA,
  fillColors = backGroundColor,
  fillTransformation = NA,
  fillColorsMinMax = NA,
  fillFunction = NA,
  showFillLegend = T,
```

```
  fillLegendTitle = fillColorColumn,
  fillLegendDirection = "horizontal",
  legendPosition = "bottom",
  legendBox = "vertical",
  naFillColor = backGroundColor,
  textSize = 4,
  textFontFace = "bold",
  linewidth = 0.35,
  alpha = 1,
  tileWidth = 1,
  tileHeight = 1,
  title = ggplot2::waiver(),
  subtitle = ggplot2::waiver(),
  caption = ggplot2::waiver()
)
```

Arguments

proteinData	the protein data data.frame
columns	numeric vector that specifies the width (in characters) of the protein data display
emptySequence	if the protein data is extended, this character vector is used to put in the sequence column in the new rows. The position column is simply filled with increasing integers.
replaceNA	default is NA, if defined as something else, then ALL NA values in the protein data will be replace with this argument. Ignored if columns is not NA.
backGroundColor	background color to use for the 'tiles' if no fill parameters are set. Default is 'white'
frontColor	color to use to border the 'tiles' in the fill color legend. If no text color parameters are set Default = 'black'
borderColor	color to use for the 'tiles'. Default is 'backGroundColor'
defaultTextColor	color to use for the text in the 'tiles' if no text color parameters are set Default = 'black'
textColorColumn	character vector: name of the column in the proteinData data.frame that contains info for the color of the text in the 'tiles'. Note: this should not be continuous data
replaceTextNA	vector that specifies with what to replace NA if there are NA values in the textColorColumn in the protein data. Default is "0" (for coverage purposes)
textColorLevels	if the data in textColor column is non-continous, eg character data, then this argument allows definition/ordering of the levels during the 'transformation' into factors (done) during visualization of the text colors

`textColorLabels`
: if the data in textColor Column is non-continous, eg character data, then this argument allows definition/ordering of the names of the levels during the 'transformation' into factors (done) during visualization of the text colors. See also `factor`

`textColors`
: vector with the text colors to use. Note: the number required is defined by the number of 'discrete' values in the textColor column of the protein data

`textColorFunction`
: allows custom definition of the function scale_color_ - function to be used during display, see example on 'fillFunction' argument

`showTextLegend`
: logical vector, if TRUE show the legend for the text color(s)

`textLegendTitle`
: title to use for the legend of the text colors

`textLegendDirection`
: direction text color legend. Default is 'vertical'. Alternative is 'horizontal'

`fillColorColumn`
: character vector: name of the column in the proteinData data.frame that contains info for the fillcolor of the 'tiles'. Note: this should be continuous data. If discrete data, then fillColorLevels can be defined for more exact coloring

`replaceFillNA`
: vector that specifies with what to replace NA if there are NA values in the fillColorColumn in the protein data

`fillColorLevels`
: if the data in fillColor column is non-continous, eg character data, then this argument allows definition/ordering of the names of the levels during the 'transformation' into factors (done) during visualization of the text colors. See also `factor`

`fillColorLabels`
: vector with the text colors to use. Note: the number required is defined by the number of 'discrete' values in the fillColor column of the protein data

`fillColors`
: the lower (minimum) and upper (maximum) bound colors to use for fillcolor of the tiles. Usually this is just two different colors: the 'tiles' will be filled with a mixture of both depending on the values in the textColorColumn

`fillTransformation`
: this allows the scale of the fill colors to be non-linear. Default is NA. One frequently used is 'log10' to generate a logarithmic scale. Other options can be found in `scale_fill_gradientn`

`fillColorsMinMax`
: the minimum and maximum value to use for forming gradient colors for the tiles. If some values in the textColorColumn are outside the range c(minimum, maximum), the some unexpected things may happen. Usually this is used to 'stretch' the color scale. Only works when working with a continuous scake

`fillFunction`
: allows for precise setting of the fill scale, see examples

`showFillLegend`
: logical vector, if TRUE show the legend for the fill color(s)

`fillLegendTitle`
: title to use for the legend of the text colors

`fillLegendDirection`
: direction fill color legend. Default is 'horizontal'. Alternative is 'vertical'

`legendPosition`
: character vector: where to put the legends ("none", "left", "right", "bottom", "top"). Default is 'bottom'

`legendBox`
: arrangement of multiple legends ("horizontal" or "vertical")

`naFillColor`
: still used ????????????? <————

`textSize`
: default is 4. Sets the size of the text in the 'tiles'

`textFontFace`
: allows manipulation of the text font used. Default is 'bold'. Alternatives are "plain", "italic", & "bold.italic"

`linewidth`
: sets the width of the lines between the 'tiles', default is 0.35

`alpha`
: sets the alpha for the fill color of the 'tiles'. Default is 1

`tileWidth`
: sets the width for the 'tiles', default is 1

`tileHeight`
: sets the height for the 'tiles', default is 1

`title`
: specifies the title

`subtitle`
: specifies the subtitle

`caption`
: specifies the caption

Value

a ggplot object

Examples

```
newTable <- proteinCoverage(sequence = standardProtein("OVA"), start = 2,
 nterm = 1, peptideTable = OVATable("peptide"), Accession = "P01012",
  positionColumn = "PositionsinProteins")
newTable <- mapPeptidesToProtein(proteinDF = newTable,
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", variable = "Abundances_1",
 dataName = "Abundances_1")
newTable <- mapPeptidesToProtein(proteinDF = newTable,
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins",
 variable = "Abundances_2", dataName = "Abundances_2")
newTable <- mapPeptidesToProtein(proteinDF = newTable,
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins",
 variable = "XCorrbySearchEngine_1", dataName = "Score")
newTable[35:40,]
# display protein sequence info
displayProtein(newTable, columns = 50)
displayProtein(newTable, columns = 50, textColorColumn = "sequence",
 textColors = rainbow(21))
displayProtein(newTable, columns = 50, textColorColumn = "sequence",
 textColors = terrain.colors(21))
# simply display sequencing coverage
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
```

```
 textColors = c("black","red"))
displayProtein(boxData(newTable, ncols = 50), textColorColumn = "coverage",
 textColors = c("black","red"))
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), backGroundColor = "lightblue")
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), textLegendTitle = "Coverage",
 showTextLegend = TRUE, textColorLabels = c("No Coverage", "Coverage"), title = "Ovalbumin")
# display score & coverage
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), textLegendTitle = "Coverage", showTextLegend = TRUE,
 textColorLabels = c("No Coverage", "Coverage"), title = "Ovalbumin",
 fillColorColumn = "Score", fillColors = c("white","brown"), alpha = 0.5)
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), textLegendTitle = "Coverage", showTextLegend = TRUE,
 textColorLabels = c("No Coverage", "Coverage"), title = "Ovalbumin", fillColorColumn = "Score",
 fillColors = c("white","brown"), alpha = 0.5, fillColorsMinMax = c(0,50))
# display abundance & coverage
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), title = "Ovalbumin", fillColorColumn = "Abundances_1",
 fillColors = c("white","brown"), alpha = 0.5)
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), title = "Ovalbumin", fillColorColumn = "Abundances_1",
 fillColors = c("white","brown"), alpha = 0.5, fillColorsMinMax = c(1E3, 1.5E10))
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), title = "Ovalbumin", fillColorColumn = "Abundances_2",
 fillColors = c("white","brown"), alpha = 0.5, fillColorsMinMax = c(1E3, 1.5E10))
# display abundance logarithmic color scale & coverage
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), title = "Ovalbumin", fillColorColumn = "Abundances_1",
 fillColors = c("white","#8b0000"), alpha = 0.5, fillColorsMinMax = c(1E6, 1.5E10),
 fillTransformation = "log10")
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), title = "Ovalbumin", fillColorColumn = "Abundances_2",
 fillColors = c("white","#8b0000"), alpha = 0.5, fillColorsMinMax = c(1E6, 1E11),
 fillTransformation = "log10")
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), title = "Ovalbumin", fillColorColumn = "Abundances_1",
 fillColors = c("white","#8b0000"), alpha = 0.5, fillColorsMinMax = c(1E6, 1.5E10),
 fillTransformation = "log10", fillLegendTitle = "Abundance", fillLegendDirection = "vertical",
 legendPosition = "right")
# use discrete data for fill can be done via 'fillFunction' argument
trypticSites <- getPeptideStart(proteinSequence = standardProtein("OVA"),
 peptideSequence = "(?<!P)R|(?<!P)K")
trypticSites <- data.frame(position = trypticSites, data = "tryptic")
trypticSites
newTable <- addDataToProtein(proteinDF = newTable, dataframe = trypticSites,
dataColumn = "data", dataName = "trypticSite", NAValue = "")
newTable[75:90,]
# w/o using the fillFunction argument
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
 textColors = c("black","red"), textLegendTitle = "Coverage", showTextLegend = TRUE,
 textColorLabels = c("No Coverage", "Coverage"), title = "Ovalbumin - tryptic sites",
```

```
 fillColorColumn = "trypticSite", fillColors = c("white","yellow"), replaceFillNA = "",
 fillColorLevels = c("","tryptic"), fillColorLabels = c("Non-tryptic site", "Tryptic site"),
 showFillLegend = FALSE, borderColor = "black")
# using the fillFunction argument
displayProtein(newTable, columns = 50, textColorColumn = "coverage",
textColors = c("black","red"), textLegendTitle = "Coverage", showTextLegend = TRUE,
textColorLabels = c("No Coverage", "Coverage"), title = "Ovalbumin - tryptic sites",
fillColorColumn = "trypticSite", fillFunction = ggplot2::scale_fill_manual(values = c("white",
"yellow"), na.value = "white"), showFillLegend = FALSE, borderColor = "black")
```

`displayProteinText` *displayProteinText*

Description

displays the protein data in a data frame as text (for console use) or HTML (for use with gt table package). Obviously more limited than the 'displayProtein' function

Usage

```
displayProteinText(
  proteinData,
  columns = NA,
  emptySequence = "",
  textColorColumn = NA,
  textColors = NA,
  textBackgroundColorColumn = NA,
  textBackgroundColors = NA,
  HTMLoutput = FALSE,
  linebreakChar = ifelse(HTMLoutput, "<br>", "\n"),
  returnText = FALSE
)
```

Arguments

`proteinData`	the protein data data.frame
`columns`	numeric vector that specifies the width (in characters) of the protein data display
`emptySequence`	character vector that specifies which value is used for empty pieces of sequence at the beginning or the end of the protein data.frame please note that these rows, with this in the sequence column, at the beginning of the data.frame are removed for display. The first non-'emptySequence' row will be the first part displayed (see also examples). If this argument is not the same as the 'emptySequence' argument used in eg `proteinCoverage`, an error will be generated
`textColorColumn`	character vector: name of the column in the proteinData data.frame that contains info for the color of the text in the 'tiles'. Note: this should not be continuous data

`textColors` vector withthe text colors to use. Note: the number required is defined by the number of 'discrete' values in the textColor column of the protein data

`textBackgroundColorColumn` character vector: name of the column in the proteinData data.frame that contains info for the backcolor of the text in the 'tiles'. Note: this should not be continuous data

`textBackgroundColors` vector with the text background colors to use. Note: the number required is defined by the number of 'discrete' values in the textBackground color column of the protein data

`HTMLoutput` logical value. Default is FALSE, if TRUE then the output will be a character vector formatted for display in an HTML environment

`linebreakChar` character vector that is used to split the resulting character vector. For console text this is '\n', for HTML output it's '< br >'. Can be adjusted to eg '< hr >' for HTML output

`returnText` default is FALSE. If no textColor and no textBackgroundColor columns are defined, then setting this to TRUE can be used to retrieve the result w/o printing it to the screen (it will contain line breaks)

Value

character vector

Note

colors for display in the console are limited. Only the ones defined by the package 'crayon' should be used. For HTML use there's (obviously) more freedom

Examples

```
newTable <- proteinCoverage(sequence = standardProtein("OVA"), start = 2, nterm = 1,
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", emptySequence = " ")
trypticSites <- getPeptideStart(proteinSequence = standardProtein("OVA"),
 peptideSequence = "(?<!P)R|(?<!P)K")
trypticSites <- data.frame(position = trypticSites, data = 1)
trypticSites
newTable <- addDataToProtein(proteinDF = newTable, dataframe = trypticSites, dataColumn = "data",
 dataName = "trypticSite", NAValue = 0)
newTable[46:55,]
# print to console
# displayProteinText(proteinData = newTable, columns = 75,
#  textColorColumn = "coverage", textColors = c("white","red"), emptySequence = " ")
# displayProteinText(proteinData = newTable, columns = 75, textColorColumn = "coverage",
#  textColors = c("white","red"), textBackgroundColorColumn = "trypticSite",
#  textBackgroundColors = c("black","yellow"), emptySequence = " ")
# HTML output via gt package
withTryptic <- displayProteinText(proteinData = newTable, columns = 75,
 textColorColumn = "coverage", textColors = c("black","red"),
 textBackgroundColorColumn = "trypticSite", textBackgroundColors = c("white","yellow"),
```

```
 emptySequence = " ", HTMLoutput = TRUE)
withoutTryptic <- displayProteinText(proteinData = newTable, columns = 75,
 textColorColumn = "coverage", textColors = c("black","red"), emptySequence = " ",
 HTMLoutput = TRUE)
different <- displayProteinText(proteinData = newTable, columns = 75,
 textColorColumn = "coverage", textColors = c("blue","green"), emptySequence = " ",
 HTMLoutput = TRUE, linebreakChar = "<hr>")
library(gt)
data.frame(Description = c("Coverage without tryptic sites", "Coverage with tryptic sites",
 "A different display..."), Sequence = c(withoutTryptic, withTryptic, different)) |>
  gt::gt() |>
  gt::fmt_markdown(columns = Sequence) |>
  gt::tab_style(
    style = list(
      gt::cell_text(font = "courier")
    ),
    locations = gt::cells_body(
      columns = "Sequence"
    )
  )
```

---

endPosition *endPosition*

Description

Gives the end position of a character vector with format 'xxx-yyy'. Convenience function

Usage

```
endPosition(string, splitCharacter = "-")
```

Arguments

string	character vector with format 'xxx-yyy', where xxx is the start position and yyy is the end position
splitCharacter	default is "-", which is used by Proteome Discoverer

Value

numeric vector

Examples

```
endPosition("901-902")
endPosition("1-102")
```

getAllModPositionInProtein *getAllModPositionInProtein*

Description

takes a data.frame with peptide information (like a peptide table from Proteome Discoverer) and gives the position(S) of a specified modification(s) (if present)

Usage

```
getAllModPositionInProtein(
  peptideTable,
  Accession,
  exact = TRUE,
  whichModification,
  positionColumn,
  clearPosition = 1L,
  unclearPosition = 2L,
  sumPositions = FALSE
)
```

Arguments

peptideTable	a data.frame with columns 'Sequence', 'Modifications' and a position column (see 'positionColumn' argument). This is what Proteome Discoverer produces
Accession	character vector that specifies from which protein Accession to get the position
exact	logical vector which defines how the Accession argument is compared to the string argument. Default is TRUE which means that the Accession needs to be exactly the same letters/numbers. If FALSE then a 'grepl' statement is used with the Accession as a pattern
whichModification	character vector that specifies for which modification the protein position is calculated (if present, name needs to be exact)
positionColumn	character vector which specifies which column contains the peptide position information. Proteome Discoverer uses either 'PositionsinProteins' or 'PositionsinMasterProteins' (depending on the settings used in the consensus method)
clearPosition	vector (default is 1L) that is used for the clear column when the position of the modification in the peptide is unambiguous (certain)
unclearPosition	vector (default is 2L) that is used when a modification position is ambiguous (uncertain)
sumPositions	logical vector which (if TRUE) causes the position information to be somewhat simplified: the clear-fields of all identical positions are summed. As an example, this causes a position which is present twice (clear and unclear) to become a single row with the clear field (clearPosition + unclearPosition)

Value

a named list of data.frame's that contain the position information

Examples

```
getAllModPositionInProtein(peptideTable = OVATable("peptide"), Accession = "P01012",
 whichModification = c("Carbamidomethyl","Phospho"), positionColumn = "PositionsinProteins")
getAllModPositionInProtein(peptideTable = OVATable("peptide"), Accession = "P01012",
 whichModification = c("Carbamidomethyl","Phospho"), positionColumn = "PositionsinProteins",
 sumPositions = TRUE)
getAllModPositionInProtein(peptideTable = OVATable("peptide"), Accession = "P01012",
 whichModification = c("Oxidation"), positionColumn = "PositionsinProteins")
getAllModPositionInProtein(peptideTable = OVATable("peptide"), Accession = "P01012",
 whichModification = c("Oxidation"), positionColumn = "PositionsinProteins", sumPositions = TRUE)
getAllModPositionInProtein(peptideTable = OVATable("peptide"), Accession = "P01012",
 whichModification = c("Something"), positionColumn = "PositionsinProteins")
```

`getModPositionInProtein` *getModPositionInProtein*

Description

takes a row of a data.frame with peptide information (like a peptide table from Proteome Discoverer) with added modification info (like coming from `modPositionsToDF`) and gives the position of a specified modification (if present)

Usage

```
getModPositionInProtein(
  peptideTableRow,
  Accession,
  exact = TRUE,
  whichModification,
  positionColumn,
  clearPosition = 1L,
  unclearPosition = 2L
)
```

Arguments

`peptideTableRow`	single row of a data.frame with a 'position column' and 'translated' modification (see examples)
`Accession`	character vector that specifies from which protein Accession to get the position

exact — logical vector which defines how the Accession argument is compared to the string argument. Default is TRUE which means that the Accession needs to be exactly the same letters/numbers. If FALSE then a 'grepl' statement is used with the Accession as a pattern

whichModification — character vector that specifies for which modification the protein position is calculated (if present, name needs to be exact)

positionColumn — character vector which specifies which column contains the peptide position information. Proteome Discoverer uses either 'PositionsinProteins' or 'PositionsinMasterProteins' (depending on the settings used in the consensus method)

clearPosition — vector (default is 1L) that is used for the clear column when the position of the modification in the peptide is unambiguous (certain)

unclearPosition — vector (default is 2L) that is used when a modification position is ambiguous (uncertain)

Value

data.frame with two columns: 'position' (position of modification in protein) and 'clear' which details if a modification position is ambiguous

Examples

```
OVATable("peptide")$Modifications
# generate data.frame of all these modifications
theModifications <- purrr::map_df(OVATable("peptide")$Modifications, ~modPositions(.x,
 returnEmptyRow = FALSE) |> modPositionsToDF(returnEmptyRow = FALSE))
theModifications |> head()
# bind together the modifications woth the original table
newTable <- dplyr::bind_cols(OVATable("peptide"), theModifications)
newTable |> head()

newTable[15,]
getModPositionInProtein(peptideTableRow = newTable[15,], Accession = "P01012",
 whichModification = "Oxidation", positionColumn = "PositionsinProteins")
newTable[32,]
getModPositionInProtein(peptideTableRow = newTable[32,], Accession = "P01012",
 whichModification = "Phospho", positionColumn = "PositionsinProteins")
getModPositionInProtein(peptideTableRow = newTable[32,], Accession = "P01012",
 whichModification = "Carbamidomethyl", positionColumn = "PositionsinProteins")
getModPositionInProtein(peptideTableRow = newTable[32,], Accession = "P01012",
 whichModification = "Carbamido", positionColumn = "PositionsinProteins")
newTable[1,]
getModPositionInProtein(peptideTableRow = newTable[1,], Accession = "P01012",
 whichModification = "Carbamidomethyl", positionColumn = "PositionsinProteins")
newTable[26,]
# # of possible positions for the phospho modification
stringr::str_count(newTable$Sequence[26], pattern = "S|T")
getModPositionInProtein(peptideTableRow = newTable[26,], Accession = "P01012",
 whichModification = "Phospho", positionColumn = "PositionsinProteins")
```

```
getModPositionInProtein(peptideTableRow = newTable[26,], Accession = "P01012",
 whichModification = "Carbamidomethyl", positionColumn = "PositionsinProteins")
# to get all Phospho positions in the protein
purrr::map_df(1:nrow(newTable), ~getModPositionInProtein(newTable[.x,], Accession = "P01012",
 whichModification = "Phospho", positionColumn = "PositionsinProteins")) |>
 dplyr::distinct(position, clear) |> dplyr::arrange(position)
```

`getPeptideStart` *getPeptideStart*

Description

Helper function to get the start positions of amino acid sequences. The function works with either a protein sequence and the peptide sequence to be searched (regular expressions can be used) or a peptide table (data.frame) containing position information in the format used by Proteome Discoverer. See also `getPositions`

Usage

```
getPeptideStart(
  peptideTable = NA,
  Accession = "",
  positionColumn,
  exact = TRUE,
  proteinSequence = NA,
  peptideSequence = NA,
  noWarnings = TRUE
)
```

Arguments

`peptideTable`	data.frame having a.position column in the Proteome Discoverer format (see `getPositions`)
`Accession`	character vector that specifies from which protein Accession to get the position
`positionColumn`	character vector which specifies which column contains the peptide position information. Proteome Discoverer uses either 'PositionsinProteins' or 'PositionsinMasterProteins' (depending on the settings used in the consensus method)
`exact`	logical vector which defines how the Accession argument is compared to the string argument. Default is TRUE which means that the Accession needs to be exactly the same letters/numbers. If FALSE then a 'grepl' statement is used with the Accession as a pattern
`proteinSequence`	character vector representing the protein sequence to be search. Ignored if peptideTable is not NA
`peptideSequence`	character vector representing the peptide sequence to search for. Regular Expressions can be used (see examples). Ignored if peptideTable is not NA

noWarnings — logical vector, default is TRUE: no warnings when a peptide sequence is present more than once in a protein sequence. If FALSE then a warning will be generated. Ignored if peptideTable is not NA

Value

numeric vector

Examples

```
# find all start positions for all peptides in the peptide table
getPeptideStart(peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins")
# find all locations of a peptide sequence
getPeptideStart(proteinSequence = standardProtein(), peptideSequence = "DILNQITK")
# find all locations of a single amino acid
getPeptideStart(proteinSequence = standardProtein(), peptideSequence = "P", noWarnings = FALSE)
# find all tryptic sites, not taking into account Proline (P)
getPeptideStart(proteinSequence = standardProtein(), peptideSequence = "K|R")
getPeptideStart(proteinSequence = standardProtein("BSA"), peptideSequence = "K|R")
# find all tryptic sites w/o a preceding Proline (P)
getPeptideStart(proteinSequence = standardProtein(), peptideSequence = "(?<!P)R|(?<!P)K")
getPeptideStart(proteinSequence = standardProtein("BSA"), peptideSequence = "(?<!P)R|(?<!P)K")
```

---

getPositionLetters *getPositionLetters*

Description

helper function which extracts the letter codes (for amino acids) from the position info part of a modification

Usage

```
getPositionLetters(position)
```

Arguments

position — character vector with format ;;... etc etc

Value

character vector with format ;;... etc etc

Note

internal function

`getPositionNumbers` *getPositionNumbers*

Description

helper function which extracts the positions from the position info part of a modification

Usage

```
getPositionNumbers(position)
```

Arguments

`position`	character vector with format ;;... etc etc

Value

character vector with format ;;... etc etc

Note

internal function

position info which does not have a letter will return a zero as position

---

`getPositions` *getPositions*

Description

Gets the part of the string containing position information. Works with data organized in the 'PositionsinProteins' and/or 'PositionsinMasterProteins' column of peptide tables as defined in Proteome Discoverer

Usage

```
getPositions(string, Accession, exact = TRUE, splitCharacter = ";")
```

Arguments

`string`	character vector in the format: 'Accession' 'position'; 'Accession' 'position'; etc
`Accession`	character vector that specifies from which protein Accession to get the position
`exact`	logical vector which defines how the Accession argument is compared to the string argument. Default is TRUE which means that the Accession needs to be exactly the same letters/numbers. If FALSE then a 'grepl' statement is used with the Accession as a pattern
`splitCharacter`	character vector which specifies the beginning/end of 'Accession' 'position' pair. Default is ";" which is used by Proteome Discoverer

Value

a character vector containing the position of the specified Accession

Note

this function expects Accessions to be unique. If this is not true or the parameter exact is set to FALSE it may return character vectors with length > 1

Examples

```
getPositions("P00000 [352-360]; P01012 [351-359]", Accession = "P01012")
getPositions("P00000 [352-360]; P01012 [351-359]", Accession = "P00000")
getPositions("P00000 [352-360]; P01012 [351-359]", Accession = "P0", exact = FALSE)
```

---

graphsAdjust *graphsAdjust*

Description

Make adjustments to a graph eg zoom, titles etc etc

Usage

```
graphsAdjust(
  graphs,
  vertical = FALSE,
  xDiscrete = FALSE,
  yDiscrete = FALSE,
  xReverse = FALSE,
  yReverse = FALSE,
  xDefault = FALSE,
  yDefault = FALSE,
  xLimits = c(0, NA),
  yLimits = c(0, NA),
  xExpand = ggplot2::expansion(mult = 0, add = 0),
  yExpand = ggplot2::expansion(mult = 0, add = 0),
  xLabelFormat = ggplot2::waiver(),
  yLabelFormat = ggplot2::waiver(),
  xOob = scales::oob_squish_infinite,
  yOob = scales::oob_squish_infinite,
  xLog = FALSE,
  yLog = FALSE,
  xIsDate = FALSE,
  yIsDate = FALSE,
  titles = NA,
  xLabel = NA,
  yLabel = NA,
```

```
  setTheme = theme_minimal_adapted(),
  plot.margins.default = TRUE,
  plot.margins = c(5, 5, 5, 5),
  plot.margins.units = "points"
)
```

Arguments

graphs	list of ggplot-objects to which the adjustments have to be made Note: MUST be a list
vertical	if TRUE, flips x- and y-axis
xDiscrete	specifies whether the x-axis should be discrete
yDiscrete	specifies whether the y-axis should be discrete
xReverse	specifies whether to reverse the x-axis
yReverse	specifies whether to reverse the y-axis
xDefault	if TRUE, then xExpand, xLimits and xOob are ignored (essentially autoscaling the x-axis)
yDefault	same as xDefault, but for y-axis
xLimits	range of the x-axis, normally xLimits = c(minimum, maximum) if minimum and/or maximum is NA, then they are autoscaled, if xLimits = NA then the range is 0, putting all datapoints in one line (x-axis-wise)
yLimits	range of the y-axis (see xLimits)
xExpand	allows for padding around data (x-axis), see ?ggplot2::expansion for proper explanation
yExpand	allows for padding around data (y-axis), see ?ggplot2::expansion for proper explanation
xLabelFormat	defines the numeric format to be used for the x-axis labels (see fromatDigits() & formatScientificDigits() for examples)
yLabelFormat	defines the numeric format to be used for the y-axis labels (see fromatDigits() & formatScientificDigits() for examples)
xOob	defines what to do with data that's out of range of the data, see ?scales::oob for proper explanation. Note: only deals with x-axis
yOob	defines what to do with data that's out of range of the data, see ?scales::oob for proper explanation. Note: only deals with y-axis
xLog	if TRUE then automatic transformation of the x-axis to logarihmic scale
yLog	if TRUE then automatic transformation of the y-axis to logarihmic scale
xIsDate	if TRUE then all settings regarding the x-axis are ignored, except xLimits which should be dates (defaults are not dates). Alternatively xDefault can be set to TRUE for autoscaling
yIsDate	if TRUE then all settings regarding the y-axis are ignored except yLimits which should be dates (defaults are not dates). Alternatively xDefault can be set to TRUE for autoscaling
titles	sets title of graph

xLabel	sets x-axis title
yLabel	set y-axos title
setTheme	if NA, then no theme is applied, otherwise uses the defined theme (can also be ggplot2 included themes, such as theme_bw())
plot.margins.default	if TRUE, ignore plot.margins and other parameters
plot.margins	defines margins (from the border) of the plot c(top, right, bottom, left)
plot.margins.units	default = "points", other possibilities: ?grid::unit , examples "cm", "points", "inches", "npc" (viewport) etc etc

Value

a list of ggplot objects

Note

also part of the personal package 'BBPersonalR'

`ifelseProper` *ifelse replacement for properly returning all datatypes.*

Description

ifelse replacement for properly returning all datatypes.

Usage

```
ifelseProper(logicValue = NULL, ifTrue = NULL, ifFalse = NULL)
```

Arguments

logicValue	variable or expression resulting in TRUE or FALSE, if missing or not logical then the function will return NULL.
ifTrue	variable or expression to be returned when logicValue == TRUE
ifFalse	variable or expression to be returned when logicValue == FALSE

Value

depending on logicValue, ifTrue ir ifFalse.

Note

not vectorized

internal function

is.Class *is.Class*

Description

helper function to determine if object == whichClass or a descendant of whichClass.

Usage

```
is.Class(object, whichClass)
```

Arguments

object	a data object of some class
whichClass	character string: class name to be tested

Value

TRUE or FALSE

Note

internal function

log10add *log10add*

Description

function factoru that generates a function that takes a numeric value, adds a specified value and then gives the logarithmic value with base 10. This is similair tp log1p. It prevents problems with zero. It does add a small value to the original value, but as this is here meant for log10 transformation of values over a large range (like Abundances of peptides in a digest), it should not be a problem.

Usage

```
log10add(valueToAdd)
```

Arguments

valueToAdd	numeric value to add to each value before taking the log10 in the resulting function

Value

function

Examples

```
log10(0:10)
log10add(1)(0:10)
log10add(0)(0:10)
log10(c(0,1E5, 1E7, 1E8))
log10add(1)(c(0,1E5, 1E7, 1E8))
log10add(1E3)(c(0,1E5, 1E7, 1E8))
```

`log10p` *log10p*

Description

takes a numeric value, adds 1 and then gives the logarithmic value with base 10. This is essentially the log10 version of `log1p`. This prevents problems with zero. It does add a small value to the original value, but as this is here meant for log10 transformation of values over a large range (like Abundances of peptides in a digest), so it should not be a problem.

Usage

```
log10p(value)
```

Arguments

`value`	numeric value

Value

numeric value

Examples

```
log10(0:10)
log10p(0:10)
log10(c(0,1E5, 1E7, 1E8))
log10p(c(0,1E5, 1E7, 1E8))
```

mapPeptidesToProtein *mapPeptidesToProtein*

Description

takes a data.frame with peptide information (like a peptide table from Proteome Discoverer) and adds data from it to the proteinData

Usage

```
mapPeptidesToProtein(
  proteinDF,
  peptideTable,
  Accession,
  positionColumn,
  shiftPosition = 0,
  variable,
  dataName = variable,
  NAValue = 0L,
  combineFunction = sum,
  na.rm = TRUE
)
```

Arguments

proteinDF	the protein data.frame to add the info to. Main requirements are that there is a column named position, that there is no column named 'newData____' (internal use) and no column with the same name as defined by the 'dataName' argument
peptideTable	a data.frame with at least a position column (see 'positionColumn' argument) and a column with the same name as the 'variable' argument. This is what Proteome Discoverer produces
Accession	character vector that specifies from which protein Accession to get the position
positionColumn	character vector which specifies which column contains the peptide position information. Proteome Discoverer uses either 'PositionsinProteins' or 'PositionsinMasterProteins' (depending on the settings used in the consensus method)
shiftPosition	integer value, if the data in the proteinDF and the data in the peptideTable are shifted, this parameter can be used to align them. Default is 0
variable	character vector: name of the column in the 'peptideTable' that needs to be mapped onto the protein data
dataName	character vector: name of the new column in the protein data data.frame
NAValue	vector to be used for positions where no peptide data maps to
combineFunction	function (default is sum) that is used for positions which have more than one peptide row mapping to it. In case of 'sum', the values are summed for that position, for the function 'max' the maximum value is taken etc etc. Note: only basic functions min, max, sum and mean have been tested

na.rm	default is TRUE. If TRUE then 'na.rm=TRUE' is added to the 'combineFunction', If FALSE, then it is ignored

Value

a data.frame

Examples

```
newTable <- createProteinData(sequence = standardProtein("OVA"), start = 2, nterm = 1, )
newTable |> head()
newTable <- mapPeptidesToProtein(proteinDF = newTable, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins", variable = "Abundances_1",
 dataName = "SummedAbundances")
newTable |> head()
newTable <- mapPeptidesToProtein(proteinDF = newTable, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins", variable = "Abundances_1",
 dataName = "Abundances", combineFunction = NA)
newTable[35:45,]
newTable[100:125,]
```

---

modPositions *modPositions*

Description

helper function which 'translates' a character vector representing a (peptide) modification to a data.frame

Usage

```
modPositions(modifications, collapseChar = ";", returnEmptyRow = TRUE)
```

Arguments

modifications	character vector representing one or more modifications. The format is 'number × modification name [position<letter(+number)>]'. If more than one modification is present, the character vector is 'combined' with separation character ';' (collapseChar)
collapseChar	character vector separating the modifications in the argument 'modifications'
returnEmptyRow	logical vector, default is TRUE. TRUE: if no modification is present in the 'modifications' argument, a data.frame will be returned without rows. FALSE: in that case a data.frame with a single row, with in every column an empty character vector, will be returned.

Value

a data.frame with columns: modifications, positions, letters and numbers

Note

if there are two or more modification present, then number of rows in the returned data.frame will also be two or more

the modification format is what Proteome Discoverer/Sequest works with in its peptide tables

Examples

```
OVATable("peptide")$Modifications[26]
modPositions(OVATable("peptide")$Modifications[26])
OVATable("peptide")$Modifications[2]
modPositions(OVATable("peptide")$Modifications[2])
OVATable("peptide")$Modifications[1]
modPositions(OVATable("peptide")$Modifications[1])
modPositions(OVATable("peptide")$Modifications[1], returnEmptyRow = FALSE)
purrr::map_df(OVATable("peptide")$Modifications, ~modPositions(.x))
```

---

modPositionsToDF *modPositionsToDF*

Description

helper function that 'collapses' two or more rows in a modification data.frane (coming from modPositions) into a single data.frame row

Usage

```
modPositionsToDF(mods, collapseChar = ",", returnEmptyRow = TRUE)
```

Arguments

mods	data.frame coming from modPositions
collapseChar	character vector (default ',') which is used to collapse the columns of the rows (needed to undo the 'collapse' in dfToModPositions)
returnEmptyRow	logical vector, default is TRUE. TRUE: if no modification is present in the 'mods' argument, a data.frame will be returned without rows. FALSE: in that case a data.frame with a single row, with in every column an empty character vector, will be returned.

Value

a data.frame with columns: modifications, positions, letters and numbers

Examples

```
OVATable("peptide")$Modifications[32]
modPositions(OVATable("peptide")$Modifications[32])
modPositions(OVATable("peptide")$Modifications[32]) |> modPositionsToDF()
OVATable("peptide")$Modifications[2]
modPositions(OVATable("peptide")$Modifications[2])
modPositions(OVATable("peptide")$Modifications[2]) |> modPositionsToDF()
```

OVATable *OVATable*

Description

Helper function for examples & testing. Generates a data.frame, of protein or peptide information (both Ovalbumin), which is similar to what comes out of Proteome Discoverer

Usage

```
OVATable(whichone = "protein")
```

Arguments

whichone	character vector which specifies what to get: 'protein' for a protein table, 'peptide' for a peptide table

Value

data.frame or NA

Examples

```
OVATable("protein")
OVATable("Peptide")
OVATable("Info")
```

---

plotProteinVariable *plotProteinVariable*

Description

creates a line plot of protein data with the amino acid position on the x-axis

Usage

```
plotProteinVariable(
  proteinDF,
  column,
  positionColumn = "position",
  addSequence = FALSE,
  sequenceColumn = "sequence",
  sequenceLevel = NA,
  textColor = "black",
  textSize = 2,
  textAlpha = 1,
  textNudgeX = 0,
  textNudgeY = 0,
  textAngle = 0,
  textFontFace = "plain",
  lineColor = "red",
  lineWidth = 0.25,
  lineType = "solid",
  lineAlpha = 1,
  drawYzero = FALSE,
  zeroColor = "blue",
  zeroWidth = 0.5,
  zeroType = "solid",
  zeroAlpha = 0.5,
  yLimits = "auto",
  yLog = FALSE,
  ylogSetValue = 1,
  incrY = ifelse(yLog, 0.5, 0.025),
  decrY = 0,
  xLimits = "auto",
  xLabel = positionColumn,
  yLabel = column,
  xLabelFormat = ggplot2::waiver(),
  yLabelFormat = ggplot2::waiver(),
  title = ggplot2::waiver(),
  subtitle = ggplot2::waiver(),
  caption = ggplot2::waiver(),
  setTheme = theme_minimal_adapted()
)
```

Arguments

proteinDF	the protein data data.frame. Obligatory columns are 'position' and 'sequence' (in case this needs to be displayed)
column	character vector: name(s) of the data column(s) that need to be displayed
positionColumn	name of the position column in the protein data which has the position info (default is 'position')

addSequence	logical vector (default FALSE). If TRUE, then the info in the argument 'sequenceColumn' is displayed in the plot
sequenceColumn	name of the column containing the sequence info, usually the amino acid sequence, tough it can be something else
sequenceLevel	defines where to display the 'sequenceColumn' info. Default is NA (ignored): sequence is displayed at the height (y-axis) of the line plot itself. Possible other values: 'min' (display at minimum y-value), 'max' (display at maximum y-value) or a numeric value (display at exactly this y-value)
textColor	color of the sequence information displayed
textSize	size of the sequence information displayed
textAlpha	alpha of the sequence information displayed
textNudgeX	offset (x-axis direction) of the sequence information displayed
textNudgeY	offset (y-axis direction) of the sequence information displayed
textAngle	number of degrees to rotate the sequence information displayed, default is 0
textFontFace	allows manipulation of the text font used. Default is 'plain'. Alternatives are "bold", "italic", & "bold.italic"
lineColor	specifies color of the line
lineWidth	specifies width of the line
lineType	specifies linetype of the line
lineAlpha	specifies alpha of the line
drawYzero	logical vector. If TRUE then a horizontal line a y = 0 is drawn/ (default is FALSE)
zeroColor	specifies color of the y=0 line
zeroWidth	specifies width of the y=0 line
zeroType	specifies linetype of the y=0 line
zeroAlpha	specifies alpha of the y=0 line
yLimits	range of the y-axis, normally yLimits = c(minimum, maximum) or 'auto' for automatic (default)
yLog	if TRUE then automatic transformation of the y-axis to logarihmic scale (default is FALSE)
ylogSetValue	in case yLog is numeric, then this value allows any y-value equal to zero or below zero to be set to this value. Default is 1. Can be set to FALSE: no change of any value
incrY	increases the upper limit of the y-axis by 100*incrY percent. Only works if yLimits is set to 'auto'
decrY	decreases the lower limit of the y-axis by 100*decrY percent. Only works if yLimits is set to 'auto'
xLimits	range of the x-axis, normally xLimits = c(minimum, maximum) or 'auto' for automatic (default)
xLabel	sets x-axis title

yLabel	set y-axos title
xLabelFormat	defines the numeric format to be used for the x-axis labels (see BBPersonalR::fromatDigits() & BBPersonalR::formatScientificDigits() for examples)
yLabelFormat	defines the numeric format to be used for the y-axis labels (see BBPersonalR::fromatDigits() & BBPersonalR::formatScientificDigits() for examples)
title	specifies the title
subtitle	specifies the subtitle
caption	specifies the caption
setTheme	specifies the theme see also ggplot2{theme}. Default is theme_minimal_adapted. If set to NA then no theme us used.

Value

a ggplot object

Examples

```
newTable <- proteinCoverage(sequence = standardProtein("OVA"), start = 2, nterm = 1,
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", emptySequence = " ")
trypticSites <- getPeptideStart(proteinSequence = standardProtein("OVA"),
 peptideSequence = "(?<!P)R|(?<!P)K")
trypticSites <- data.frame(position = trypticSites, data = 1)
trypticSites |> head()
newTable <- addDataToProtein(proteinDF = newTable, dataframe = trypticSites,
 dataColumn = "data", dataName = "trypticSite", NAValue = 0)
newTable <- mapPeptidesToProtein(proteinDF = newTable, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins",
 variable = "XCorrbySearchEngine_1", combineFunction = max)
newTable <- mapPeptidesToProtein(proteinDF = newTable, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins", variable = "Abundances_1")
newTable <- mapPeptidesToProtein(proteinDF = newTable, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins", variable = "Abundances_2")
newTable[41:50,]
plotProteinVariable(proteinDF = newTable, column = "XCorrbySearchEngine_1",
 yLabel = "Score")
plotProteinVariable(proteinDF = newTable, column = "XCorrbySearchEngine_1",
 yLabel = "Score", addSequence = TRUE)
plotProteinVariable(proteinDF = newTable, column = "XCorrbySearchEngine_1",
 yLabel = "Score", addSequence = TRUE, textColor = "blue", xLimits = c(51,100),
 yLimits = c(-0.1, 11.1))
plotProteinVariable(proteinDF = newTable, column = "XCorrbySearchEngine_1", yLabel = "Score",
 addSequence = TRUE,  textColor = "blue", sequenceLevel = "min", xLimits = c(51,100),
  yLimits = c(-0.1, 11.1))
plotProteinVariable(proteinDF = newTable, column = "XCorrbySearchEngine_1",
 yLabel = "Score", addSequence = TRUE,  textColor = "blue", sequenceLevel = 5,
 xLimits = c(51,100), yLimits = c(-0.1, 11.1))
plotProteinVariable(proteinDF = newTable, column = "Abundances_1", yLabel = "Abundance")
plotProteinVariable(proteinDF = newTable, column = "Abundances_1", yLabel = "Abundance",
 yLog = TRUE, ylogSetValue = FALSE, yLimits = c(1E3, 2E10), drawYzero = TRUE,
```

```
 zeroType = "dotted", zeroAlpha = 0.5)
plotProteinVariable(proteinDF = newTable, column = "Abundances_1",
 yLabel = "Abundance", yLog = TRUE, ylogSetValue = 1E4, yLimits = c(1E3, 2E10),
 drawYzero = TRUE, zeroType = "dotted", zeroAlpha = 0.5)
plotProteinVariable(proteinDF = newTable, column = c("Abundances_1", "Abundances_2"),
 lineColor = c("blue", "red"), yLabel = "Abundance", yLog = TRUE, ylogSetValue = 1E4,
 yLimits = c(1E3, 2E10), drawYzero = TRUE, zeroType = "dotted", zeroAlpha = 0.5)
```

---

prepare *prepare*

Description

Helper function to add (empty) rows to proteinData,

Usage

```
prepare(
  data,
  start = min(data$position),
  end = max(data$position),
  emptySequence = " "
)
```

Arguments

data	data.frame with two columns: position (integer vector) and sequence (amino acid letters: character vector)
start	integer, should be less than the start position (in first row of data): adds positions with 'emptySequence' to the head of data
end	integer, should be maximum positione: adds positions with 'emptySequence' to the tail of data
emptySequence	default " ", character vector for the sequence column to use when adding rows. Note: other columns will also be extended and usually add NA's as a value

Value

data.frame with two columns: position (integer vector) and sequence (amino acid letters: character vector)

Examples

```
createProteinData(sequence = standardProtein("bsa"), start = 25) |> prepare(start = 1) |> head()
createProteinData(sequence = standardProtein("bsa"), start = 25) |> prepare(end = 610) |> tail()
```

proteinCoverage *proteinCoverage*

Description

Create a protein data.frame with basic information: position, sequence & coverage. In the resulting data.frame, every row has an amino acid (letter, sequence column), the position of that amino acid in the protein (position column) and a coverage column where 0 is no coverage and 1 is coverage.

The function can work with a protein table row (as in Proteome Discoverer results), which should contain the Sequence and the SequenceCoverage columns. An alternative is using a sequence character vector containing a protein amino acid sequence and a peptide table (as in Proteome Discoverer results) of that same sequence (protein).

Usage

```
proteinCoverage(
  proteinTableRow = NA,
  checkSequenceLength = TRUE,
  noWarnings = FALSE,
  prepareColumns = NA,
  prepareRows = NA,
  emptySequence = "",
  noCoverage = 0,
  coverage = 1,
  sequence = NA,
  start = 1,
  end = ifelse(is.na(sequence), 1, nchar(sequence)),
  nterm = NA,
  cterm = NA,
  peptideTable,
  Accession,
  positionColumn,
  shiftPosition = 0
)
```

Arguments

proteinTableRow

data.frame row that has at least a Sequence column with the whole protein amino acid sequence and a SequenceCoverage column specifying the coverage. The SequenceCoverage needs to be in format: 'full sequence length';'start-end 1st part';'start-end 2nd part'; etc etc The start-end parts together specify which parts of the protein have been sequenced (coverage). Note: the positions SequenceCoverage field in the proteinTableow are zero based: the first amino acid position is zero, not one! This behavior is consistent with protein tables as used by Proteome Discoverer and can be changed by the proteinTableRowZero argument.

checkSequenceLength
: logical vector: default is TRUE, defined for odd situations where there may be malformed SequenceCoverage fields. Might be removed in future versions

noWarnings
: logical vector: default is FALSE. If the check for the sequence length fails, it will generate a warning. If TRUE then no warning is generated. Might be removed in future versions

prepareColumns
: defines the number of columns for a 2D matrix (puts the amino acid letters in a rows*columns matrix)

prepareRows
: defines the number of rows for a 2D matrix (puts the amino acid letters in a rows*columns matrix)

emptySequence
: default "-", character vector for the sequence column to use when adding rows. Note: other columns will also be extended and usually add NA's as a value. Ignored when proteinTableRow is not NA.

noCoverage
: defines what to use when a position is not 'covered'. Can be a number, logical or character. It will be changed into a character vector at the end of the function. Default is 0 (integer)

coverage
: defines what to use when a position is 'covered'. Can be a number, logical or character. It will be changed into a character vector at the end of the function. Default is 1 (integer)

sequence
: character vector, protein sequence in the form of the usual letter codes. Ignored when proteinTableRow is not NA.

start
: first position in the sequence to be used. Ignored when proteinTableRow is not NA.

end
: last position in the sequence to be used. Ignored when proteinTableRow is not NA.

nterm
: start position 'adjustment'. Adds a number of 'empty' rows to the head of the resulting data.frame to have it start at position nterm. Ignored when proteinTableRow is not NA.

cterm
: end position 'adjustment'. Adds a number of 'empty' rows to the tail of the resulting data.frame to have it ends at position cterm

peptideTable
: data.frame containing the peptide information to be mapped to the sequence. Sholud contain either a column named 'PositionsinProteins' or 'PositionsinMasterProteins'. This column should be character vectors of one or more of following element: [start:end]; Ignored when proteinTableRow is not NA.

Accession
: character vector that specifies from which protein Accession to get the position. This is needed for when using the peptideTable as a peptide may be present in more than one protein. Ignored when proteinTableRow is not NA.

positionColumn
: character vector which specifies which column contains the peptide position information. Proteome Discoverer uses either 'PositionsinProteins' or 'PositionsinMasterProteins' (depending on the settings used in the consensus method)

shiftPosition
: integer value, if the data in the protein sequence and the data in the peptideTable are shifted, this parameter can be used to align them. Default is 0 and ignored when proteinTableRow is not NA.

Value

data.frame with same format as output by createProteinData or boxData with the 'coverage' column added. The values in the coverage column are either 0 (no coverage) or 1 (coverage)

Note

it is possible to use both prepareColumns and prepareRows at the same time, but this may lead to errors and unexpected results. It is therefore not recommended to do so.

Examples

```
proteinCoverage(OVATable())
proteinCoverage(OVATable(), prepareColumns = 50)
proteinCoverage(OVATable()) |> boxData(ncols = 50)
proteinCoverage(sequence = substr(standardProtein(), 2, 386), peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins")
```

---

reduceColors *reduceColors*

Description

takes protein data with color information and creates a smaller data.frame contaning three columns: start, end (both positions) and color. This can be used for mapping the colors onto the 3D object. It is used inside the

Usage

```
reduceColors(proteinDF, positionColumn = "position", colorColumn)
```

Arguments

proteinDF	protein data data.frame
positionColumn	name of the column containing the position information
colorColumn	name of the column containing the color information

Value

data.frame with three columns: start, end (both positions) and color

Note

internal function

`rowsAndcolumns` *rowAndColumns*

Description

Calculates number of columns and rows of a matrix from number of cells and the ratio number of columns/ number of rows The number of cells is a defined as a minimum number of cells. Either the number of rows or the number of columns is increased to accommodate this. The column/row ratio and number of cells for the 'solution' might therefore not be exact. The reason for this is that the intended target for this function is to come up with a cell matrix big enough for a data of a certain size (vectorLength)

Usage

```
rowsAndcolumns(
  vectorLength,
  ratio,
  increaseColumns = FALSE,
  minimumRatio = 0.01,
  maximumRatio = 100
)
```

Arguments

`vectorLength`	minimum number of cells for the resulting columns*rows matrix
`ratio`	the intended ratio the resulting columns*rows matrix. Increase ratio to get more columns, decrease to get more rows. Note: ratio = number of columns divided by the number of rows.
`increaseColumns`	if TRUE, then the number of columns will be increased to reach the number of matrix cells required. If FALSE then the number of rows will be increased to do this. Default is FALSE
`minimumRatio`	due to the iterative nature of part of the function, this puts a minimum limit on the possible ratios to prevent the function from 'getting stuck'. Default is 0.01
`maximumRatio`	due to the iterative nature of part of the function, this puts a maximum limit on the possible ratios to prevent the function from 'getting stuck'. Default is 100

Value

a named numeric vector (names: columns, rows)

Examples

```
rowsAndcolumns(385, 5)
rowsAndcolumns(385, 0.2)
rowsAndcolumns(400, 10)
rowsAndcolumns(400, 0.1)
```

`standardProtein` *standardProtein*

Description

Helper function for examples & testing. Generate a character vector of the amino acid sequence of one of the (2) standard proteins.

Usage

```
standardProtein(short = "OVA")
```

Arguments

`short`	shortened name of the protein to retrieve. Only 'bsa' and 'ova' (or their upper case equivalents) are valid inputs. Other input will return NA

Value

an amino acid sequence in the form of a character vector or NA

Note

The data is coming from UniProt

OVA is short for P01012 · OVAL_CHICK Ovalbumin (Gallus gallus)

BSA is short for P02769 · ALBU_BOVIN Bovine Serum Albumin (Bos taurus)

Examples

```
standardProtein()
standardProtein("bsa")
standardProtein("Insulin")
```

`startPosition` *startPosition*

Description

Gives the start position of a character vector with format 'xxx-yyy'. Convenience function

Usage

```
startPosition(string, splitCharacter = "-")
```

Arguments

string	character vector with format 'xxx-yyy', where xxx is the start position and yyy is the end position
splitCharacter	default is "-", which is used by Proteome Discoverer

Value

numeric vector

Examples

```
startPosition("901-902")
startPosition("1-102")
```

---

strMultiReplaceAll *strMultiReplaceAll*

Description

helper function that replaces multiple patterns in a character vector, essentially an extension of stringr::str_replace_all

Usage

```
strMultiReplaceAll(strings, patterns, replacements = "")
```

Arguments

strings	character vector (can be longer than 1) in which all the patterns (arguments) are to be replaced
patterns	character vector of patterns to be replaced
replacements	character vector of replacements for the patterns in the strings argument (if found). Should be either length = 1 or same length as the replacements. Default is "" (empty character vector), which means that all patterns in strings will be removed

Value

character vector

Note

internal function

`theme_minimal_adapted` *theme_minimal_adapted*

Description

to be able to use the theme theme_minimal with some adjustments

Usage

```
theme_minimal_adapted(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  axis_line_size = base_line_size/2,
  xAxis = TRUE,
  yAxis = TRUE,
  showLegend = TRUE,
  legend.position = "bottom",
  gridLines = TRUE,
  gridLinesX = TRUE,
  gridLinesY = TRUE,
  titleSize = NA
)
```

Arguments

`base_size`	size of the lettering of axis, title etc
`base_family`	letter type of axis, title etc
`base_line_size`	width of gridlines, set to 0 for none
`axis_line_size`	width of axis lines, set to 0 for none
`xAxis`	if TRUE then display xAxis title
`yAxis`	if TRUE then display yAxis title
`showLegend`	if TRUE then show legend
`legend.position`	defines where to place the legend
`gridLines`	if TRUE then display gridlines
`gridLinesX`	if TRUE then display gridlines 'along' the x-axis
`gridLinesY`	if TRUE then display gridlines 'along' the y-axis
`titleSize`	if NA, use default title size, else use titleSize value
	To be used as ggplot-object + theme_minimal_adapted()

Value

theme definition

Note

also part of the personal package 'BBPersonalR'

`translateRangesToColors` *translateRangesToColors*

Description

more advanced function that takes a data.frame of information and a protein data data.frame and generates colors matching the instructions in the information data.frame applied to the protein data.frame. This allows for more complicated coloring schemes

Usage

```
translateRangesToColors(ranges, proteinData, outsideColor)
```

Arguments

`ranges`	data.frame with 4 columns: 'rangeValues', 'insideColor', 'inbetween' (similair in use to in `translateRangeToColors`) & 'columnName' which determines from which column, in the protein data data.frame, values should be taken to determine the colors. Note: that there can be colors in the earlier rows of this data.frame will be overwritten if needed by colors in later rows if needed
`proteinData`	a proteinData data.frame with at least the columns specified in the ranges data.frame
`outsideColor`	color to apply to any row in the proteinData that falls outside any of the ranges

Value

a vector of colors with the same length as the number of rows in the protein data

Examples

```
# create protein data
ovaProtein <- proteinCoverage(sequence = standardProtein(),
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", shiftPosition = 1)
# create ranges data.frame
dfColors <- data.frame(rangeValues = NA,
insideColors = c("#0053D6", "#65CBF3", "#FFDB13", "#FF7D45"),
columnName = "position", inbetween = FALSE)
dfColors$rangeValues <- list(c(1:10), c(11:20), c(21:30), c(31:40))
dfColors
translateRangesToColors(ranges = dfColors, proteinData = ovaProtein,
 outsideColor = "white")[1:45]
# create ranges data.frame
dfColors <- data.frame(rangeValues = NA,
insideColors = c("#0053D6", "#65CBF3", "#FFDB13", "#FF7D45"),
```

```
columnName = "sequence", inbetween = FALSE)
dfColors$rangeValues <- list(c("A","C","D"), c("E","F"), c("G","H"), c("K","L"))
dfColors
translateRangesToColors(ranges = dfColors, proteinData = ovaProtein,
 outsideColor = "white")[1:45]
```

---

translateRangeToColors *translateRangeColors*

Description

function factory that generates a color translation function which assigns a color based on whether the argument value is between two values or if the argument value is part of a range values

Usage

```
translateRangeToColors(
  rangeValues,
  insideColor,
  outsideColor = NA,
  inbetween = TRUE
)
```

Arguments

rangeValues	numeric vector, range of values: if 'inbetween' is TRUE, then the resulting function will determine if a value falls in between the first two values. If 'inbetween' is FALSE, then the resulting function will check if a value is part of the rangeValues argument. Note that in that case one can work with non-numeric values, see example#'
insideColor	color to assign to values that fall within or are part of a range
outsideColor	color to assign to values that fall outside or are not part of a range
inbetween	logical value that determines the type of range checking

Value

a function that takes the argument 'value', which it then translates to the correct color (vectorized)

Examples

```
translateRangeToColors(rangeValues = c(1, 10), insideColor = "white")(1:20)
translateRangeToColors(rangeValues = c(1, 10), insideColor = "white", inbetween = FALSE)(1:20)
translateRangeToColors(rangeValues = 1:10, insideColor = "white", inbetween = FALSE)(1:20)
translateRangeToColors(rangeValues = c(1,2,3, 11, 12, 13), insideColor = "white",
 outsideColor = "red", inbetween = FALSE)(1:20)
translateRangeToColors(rangeValues = c("A","B","C"), insideColor = "white",
 outsideColor = "red", inbetween = FALSE)(LETTERS)
```

translateToFixedColors *translateToFixedColors*

Description

function factory that generates a function that 'tramslates' (maps) protein data to colors. Meant for non-continous data. If the colors are then used as a columnn for the protein data data.frame, it can be used modify colors on a 3d protein object. See addStyleElement for more info'

Usage

```
translateToFixedColors(translateDF, notPresentColor)
```

Arguments

translateDF	data.frame with two columns: one has the different data elements that need to be translated to a certain color. The second column should be named 'color' and should contain the colors to which the values in the first column are translated. Examples: data.frame(position = 1:10, color = c("blue","red")) or data.frame(residue = c("C", "A", "P"), color = c("yellow", "red", "blue"))
notPresentColor	color(code) for when a value is not in the translation data.frame

Value

function that takes the argument 'value', which it then translates the argument to its corresponding color(code)s. The function is vectorized so the argument can be multi-element

Examples

```
# create protein data
ovaProtein <- proteinCoverage(sequence = standardProtein(),
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", shiftPosition = 1)
# add tryptic sites
ovaProtein <- addDataToProtein(proteinDF = ovaProtein,
 dataframe = data.frame(position = getPeptideStart(proteinSequence = standardProtein(),
 peptideSequence = "(?<!P)R|(?<!P)K"), data = "tryptic"), dataColumn = "data",
 dataName = "realTrypticSite", NAValue = "non tryptic")
# add colors for the tryptic sites
ovaProtein$trypticColor <- translateToFixedColors(translateDF =
 data.frame(position = c("tryptic"), color = "red"),
 notPresentColor = "white")(ovaProtein$realTrypticSite)
# add colors for proline positions
ovaProtein$proline <- translateToFixedColors(translateDF = data.frame(sequence = "P",
 color = "red"), notPresentColor = "white")(ovaProtein$sequence)
ovaProtein |> dplyr::slice(15:20)
```

translateToGradientColors *translateToGradientColors*

Description

function factory that generates a function that 'tramslates' (maps) protein data to colors. Meant for continous data. If the colors are then used as a columnn for the protein data data.frame, it can be used modify colors on a 3d protein object. See addStyleElement for more info'

Usage

```
translateToGradientColors(
  minValue,
  maxValue,
  colorSteps = 1000,
  colors,
  transformFunction = NA
)
```

Arguments

minValue	minimum value to use when translating to colors. If a value is lower than this, it will be set to this
maxValue	maximum value to use when translating to colors. If a value is higher than thism it will be set to this
colorSteps	essentially the resolution of the gradient. If set to 1000 (default), then a gradient of 1000 individual colors will be created to translate to. Do not set excessively high or low
colors	two element character vector. Specifies the colors for the minimumValue amd the maximumValue. The translation will have 'colorSteps' gradations between these two colors
transformFunction	allows for passing a transformation function to the values to be translated. The minValue and maxValue arguments will also be transformed. This can be used where the value-range is (very) large

Value

function that takes the argument 'value', which is then used to translate the argument to the corresponding color(code)s. The function is vectorized so the argument can be multi-element. Note: the 'transformFunction' specified is applied to the argument!

Examples

```
# create protein data
ovaProtein <- proteinCoverage(sequence = standardProtein(),
 peptideTable = OVATable("peptide"), Accession = "P01012",
 positionColumn = "PositionsinProteins", shiftPosition = 1)
# mapping Abundance information
ovaProtein <- mapPeptidesToProtein(proteinDF = ovaProtein, peptideTable = OVATable("peptide"),
 Accession = "P01012", positionColumn = "PositionsinProteins",
 shiftPosition = 1L, variable = "Abundances_1",
 dataName = "Abundance", combineFunction = sum)
# create the color translation function
linearColors <- translateToGradientColors(minValue = min(ovaProtein$Abundance),
 maxValue = max(ovaProtein$Abundance),
 colorSteps = 1000, colors = c("white", "blue"))
linearColors(c(0, 1E5, 1E9))
# create a logarithmic translation function
logColors <- translateToGradientColors(minValue = 1E3,
 maxValue = 1E10,
 colorSteps = 1000, colors = c("white", "blue"), transformFunction = log10p)
logColors(c(0, 1E5, 1E9))
# this doesn't work, due to log10 not generating NA's
logColors <- translateToGradientColors(minValue = min(ovaProtein$Abundance),
 maxValue = max(ovaProtein$Abundance),
 colorSteps = 1000, colors = c("white", "blue"), transformFunction = log10)
logColors(c(0, 1E5, 1E9))
# this does work
logColors <- translateToGradientColors(minValue = min(ovaProtein$Abundance),
 maxValue = max(ovaProtein$Abundance),
 colorSteps = 1000, colors = c("white", "blue"), transformFunction = log10p)
logColors(c(0, 1E5, 1E9))
# limit the value range
logColors <- translateToGradientColors(minValue = 1E3,
 maxValue = max(ovaProtein$Abundance),
 colorSteps = 1000, colors = c("white", "blue"), transformFunction = log10)
logColors(c(0, 1E5, 1E9))
# same result in a different way
logColors <- translateToGradientColors(minValue = min(ovaProtein$Abundance),
 maxValue = max(ovaProtein$Abundance),
 colorSteps = 1000, colors = c("white", "blue"), transformFunction = log10add(1E3))
logColors(c(0, 1E5, 1E9))
# add the colors to the protein data
ovaProtein$AbundanceColors <- linearColors(ovaProtein$Abundance)
ovaProtein$AbundanceLogColors <- logColors(ovaProtein$Abundance)
ovaProtein |> dplyr::slice(35:45)
ovaProtein |> dplyr::slice(360:375)
```

`whichModifications` *whichModifications*

Description

Helper function to list modifications in a character vector of format modification,splitCahracter,modification,splitCahracter, etc etc

Usage

```
whichModifications(modification, splitCharacter = ",", trim = TRUE)
```

Arguments

modification	character vector of format etc etc
splitCharacter	character vector used to split the modification argument
trim	logical vector. Default is TRUE: the result will be trimmed (spaces on both sides removed). When FALSE, this is not done

Value

character vector

Examples

```
whichModifications('Oxidation,Phospho')
whichModifications('Oxidation , Phospho')
whichModifications('Oxidation , Phospho', trim = FALSE)
whichModifications('Oxidation , Phospho, Carbamidomethyl')
whichModifications(c('Oxidation , Phospho','Carbamidomethyl, Phospho'))
```

Index