Package: massSpectrometryR (via r-universe)

October 26, 2024

Type Package

Imports R6, stringr, magrittr, tibble, dplyr, rcdk, purrr, enviPat

Title massSpectrometryR

Version 0.3.0

Author Ben Bruyneel <benbruyneel@gmail.com>

Maintainer Ben Bruyneel <benbruyneel@gmail.com>

Description Provides calculations, plotting etc for chemistry & mass spectrometry.

License GPL (>= 3)

Encoding UTF-8

URL https://github.com/BenBruyneel/massSpectrometryR

LazyData true

RoxygenNote 7.2.3

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Depends R (>= 2.10)

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

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

RemoteRef HEAD

RemoteSha 066d41a966bb54213fa155e8149a10491326958c

Contents

`addFormulas` *Adding up two formulas, taking into account possible differing elements*

Description

Adding up two formulas, taking into account possible differing elements

Usage

```
addFormulas(formula1, formula2)
```

Arguments

formula1	named numeric vector, example c(O = 2, C = 1)
formula2	named numeric vector, example c(H = 2, S = 1)

Value

a named numeric vector (formula)

Examples

```
addFormulas(waterFormula(), protonFormula())
addFormulas(waterFormula(), c(C=1, O=2))
```

---

addListFormulas *Add up a list of formulas*

Description

Take a list of formulas and adds them all up

Usage

```
addListFormulas(formulas)
```

Arguments

formulas	list of formulas

Value

a named numeric vector (formula)

Examples

```
addListFormulas(list(c(H = 2, O = 1),
                     c(H = 1),
                     c(H = 2, O = 1),
                     c(S = 1, O = 2)))
```

aminoAcidClass *R6 Class representing a set of amino acids*

Description

R6 Class representing a set of amino acids. It adds three functions to quickly switch between different writing 'styles' of peptides

Details

Note: this class is meant to be used only for amino acids and such

Super class

massSpectrometryR::chemicals -> aminoacids

Methods

Public methods:

- aminoAcidClass$getName()
- aminoAcidClass$getShort()
- aminoAcidClass$translatePeptide()
- aminoAcidClass$clone()

Method `getName()`**:** Function to retrieve the full name of an amino acid via the letter or shorts

Usage:

```
aminoAcidClass$getName(searchString, checkCase = TRUE)
```

Arguments:

`searchString` either a 1- or 3- letter character vector

`checkCase` default = TRUE. If false, the function will ignore the case the searchString argument

Returns: character vector, name of the aminoacid

Method `getShort()`**:** Function to retrieve either the 1- or 3- letter code of an amino acid

Usage:

```
aminoAcidClass$getShort(searchString, checkCase = TRUE)
```

Arguments:

`searchString` either a 1- or 3- letter character vector: if 1-letter than the corresponding 3-letter character vector will be returned and vice versa

`checkCase` default = TRUE. If false, the function will ignore the case the searchString argument

Returns: character vector, 1- or 3- letter code of the aminoacid

Method `translatePeptide()`**:** Translates a amino acid sequence from 1-letter codes to 3-letter codes and vice versa

Usage:

```
aminoAcidClass$translatePeptide(
  sequence,
  from1to3 = FALSE,
  splitCharacter = NA,
  joinCharacter = NA,
  checkCase = TRUE
)
```

Arguments:

`sequence` character vector: amino acid sequence in 1-letter or 3-letter codes

`from1to3` logical vector: if TRUE, then translation will be from 1-letter code to 3=letter code. If FALSE, then vice versa. Default = FALSE

`splitCharacter` character vector specifying the character(s) between the 1- or 3-letter codes in the sequence. Default NA (same as "")

`joinCharacter` character vector specifying the character(s) between the translated codes. Default NA (same as "")

`checkCase` default = TRUE. If false, the function will ignore the case the sequence

Returns: character vector, sequence in either 1- or 3-letter codes

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
aminoAcidClass$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
aminoAcidResidues()$getShort("L")
aminoAcidResidues()$getShort("Leu")
aminoAcidResidues()$getName("L")
aminoAcidResidues()$getName("Leu")
aminoAcidResidues()$translatePeptide("Asp-Arg-Val-Tyr-Ile-His-Pro-Phe-His-Leu",
 from1to3 = TRUE, splitCharacter ="-")
aminoAcidResidues()$translatePeptide("DRVYIHPFHL", joinCharacter = "-")
```

`aminoAcidModifications`

Returns a pre-defined object which contains info on some common amino acid modifications

---

Description

Returns a pre-defined object which contains info on some common amino acid modifications

Usage

```
aminoAcidModifications()
```

Value

An object of class modifications containing info on amino acid modifications

Note

the resulting modification table cannot be used immediately: there is two times a fixed modification for Cysteine amino acids. Remove one of them to prevent errors when using peptide calculations

Examples

```
print(aminoAcidModifications)
```

`aminoAcidResidues` *Generates a pre-defined object which contains info on 'normal' amino acid residues*

Description

Generates a pre-defined object which contains info on 'normal' amino acid residues

Usage

```
aminoAcidResidues()
```

Value

a R6 object of class 'chemicals'

Note

The formulas in the object are amino acid residues as they are present in proteins. To get the actual formula of the amino acid in its 'free' form, add c(H=2, O=1) (water)

this object is used in all protein calculations in this package

Examples

```
print(aminoAcidResidues())
```

`chemicals` *R6 Class representing a set of chemicals*

Description

Every chemical inside the object has a name, letter, short and a formula. The first 3 can be any length of string (though the letter and short field should be maximum length (nchar) 1 and 2-4 respectively). Formula should be in the form of a named numeric with the names representing elements and the values themselves being the number of atoms of that element, eg c(C = 3, H = 5, N = 1, O = 1, S = 0)

Details

Note: this class is meant to be used for classes of compounds, eg amino acids

Also: his class is meant as a base class to be expanded via inheritance

Warning: all chemicals inside this object should be unique (names, letters & shorts)

Active bindings

`number` retrieve the number of compounds present in the object, read only

`letters` to access the letters of the compounds in the object

`names` to access the names of the compounds in the object

`shorts` to access the shorts of the compounds in the object

`formulas` to access the formulas of the compounds in the object

`table` retrieves all info on the compounds in data.frame format, read only

Methods

Public methods:

- `chemicals$new()`
- `chemicals$print()`
- `chemicals$getFormula()`
- `chemicals$clone()`

Method `new()`**:** Create a new chemicals object

Usage:

```
chemicals$new(letters, shorts, names, formulas)
```

Arguments:

`letters` character vector specifying the letters (or numbers or whatever) for the chemicals. In case of amino acids it should be eg "A" for Alanine, "G" for Glycine, etc etc

`shorts` character vector specifying the short names for the chemicals, eg Ala for Alanine

`names` character vector specifying the names of the chemicals

`formulas` list of named numeric vectors specifying the formulas of the chemicals, eg c(C = 6, H = 12, N = 4, O = 1, S = 0) for Arginine

Returns: a new 'chemical' object

Method `print()`**:** For printing purposes: prints a table of the chemicals with columns letter, name & short

Usage:

```
chemicals$print(...)
```

Arguments:

`...` no arguments, the function takes care of printing

Method `getFormula()`**:** Retrieves the formula of one of the compounds in the object

Usage:

```
chemicals$getFormula(which1)
```

Arguments:

`which1` specifies which chemical should be retrieved. Which the number (row number in the chemicals table), or the name, letter or short as a character vector. The way this is set up, it doesn't matter whether capital or non-capital letters are used, since all is converted to upper case before comparing with what's in the chemical table

Returns: a formula in the shape of a named numeric vector, eg c(C = 6, H = 12, N = 4, O = 1, S = 0)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
chemicals$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
estrogens <- chemicals$new(letters = c("1","2","3","4"),
                           shorts = c("E1","E2","E3","E4"),
                           names = c("Estrone","Estradiol",
                                     "Estriol","Estetrol"),
                           formulas = list(c(C=18, H=22, O=2),
                                           c(C=18, H=24, O=2),
                                           c(C=18, H=24, O=3),
                                           c(C=18, H=24, O=4)))
```

digest *Digests a sequence and returns*

Description

Digests a sequence and returns

Usage

```
digest(sequence, enzyme = "trypsin", missed = 0)
```

Arguments

sequence	character vector representing the amino acid sequence to be digested. Note: the letters in sequence will be changed to upper case.
enzyme	character string specifying the enzyme to be used for the digestion. Default is 'trypsin'. Other options are 'trypsin.strict', 'pepsin', 'chymotrypsin' and 'chymotrypsin.strict'
missed	integer vector: the maximum number of allowed missed cleavages

Value

data.frame with the columns 'peptide', 'start', 'stop' and 'mc' (missed cleavages)

Note

This function is an modified version of the Digest function found in the package 'OrgMassSpecR'

electronFormula *generates a pre-defined formula for electron*

Description

generates a pre-defined formula for electron

Usage

```
electronFormula()
```

Value

a named numeric vector (formula)

Note

this is used for calculations

Examples

```
print(electronFormula())
```

`elements` *R6 Class representing a set of elements*

Description

Every element inside the object has a name, letter and a mass. The first 2 can be any length of string, mass should be a numeric

Details

Note: this class is meant to be used for elements.

Warning: all elements inside this object should be unique (names & shorts, not mass)

Active bindings

`number` retrieve the number of elements present in the object, read only

`names` to access the names of the elements in the object

`shorts` to access the shorts of the elements in the object

`mass` to access the masses of the elements in the object

`table` retrieves all info on the elements in data.frame format, read only

Methods

Public methods:

- `elements$new()`
- `elements$addElement()`
- `elements$print()`
- `elements$getMass()`
- `elements$clone()`

Method `new()`**:** creates a new elements object

Usage:

```
elements$new(shorts, names, mass)
```

Arguments:

`shorts` character vector specifying the short names for the elements, eg Hg for Mercury

`names` character vector specifying the names of the elements

`mass` numeric vector specifying the masses of the elements

Returns: a new 'elements' object

Method `addElement()`**:** adds one or more elements to the object. Elements to be added must have unique names and shorts.

Usage:

```
elements$addElement(shorts, names, mass)
```

Arguments:

`shorts` character vector specifying the short names for the elements, eg Hg for Mercury

`names` character vector specifying the names of the elements

`mass` numeric vector specifying the masses of the elements

Returns: nothing

Method `print()`**:** For printing purposes: prints a table of the chemicals with columns letter, name & short

Usage:

```
elements$print(...)
```

Arguments:

`...` no arguments, the function takes care of printing

Method `getMass()`**:** Retrieves the mass of one of the elements in the object

Usage:

```
elements$getMass(which1)
```

Arguments:

`which1` specifies which element should be retrieved. Which number (row number in the chemicals table), name or short as a character vector. The way this is set up, it doesn't matter whether capital or non-capital letters are used, since all are converted to upper case before comparing with what's in the elements table

Returns: a numeric value

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
elements$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
randomElements <- elements$new(shorts = c("X1","X2","X3"),
                               names  = c("Secret Element 1",
                                          "Secret Element 2",
                                          "Secret Element 1"),
                               mass   = c(301, 312, 323))
```

`elementsAverage` *generates a pre-defined object which contains info on elements, mass values are average masses (weighted mean mass of elements based on their natural occurence)*

Description

generates a pre-defined object which contains info on elements, mass values are average masses (weighted mean mass of elements based on their natural occurence)

Usage

```
elementsAverage()
```

Value

an elements object

Note

electron is not really meant to be used in formulas, but is needed to calculate mass & m/z of ions

Examples

```
print(elementsAverage())
```

`elementsInFormula` *elementsInFormula*

Description

retrieves the names of the elements in a formula

Usage

```
elementsInFormula(formula, removeZero = FALSE)
```

Arguments

`formula`	named numeric vector, example c(O = 2, C = 1)
`removeZero`	logical flag on how to deal with elements which are zero

Value

character vector

Examples

```
glucose = c(C=6, H=12, O=6, S=0)
elementsInFormula(glucose)
elementsInFormula(glucose, removeZero = TRUE)
```

---

elementsInFormulas *Combine elements present in 2 separate formulas*

Description

especially important when formula1 contains elements that formula2 does not contain and vice versa, Note: also sorts the result

Usage

```
elementsInFormulas(formula1, formula2, decrease = FALSE)
```

Arguments

formula1	named numeric vector, example c(O = 2, C = 1)
formula2	named numeric vector, example c(H = 2, S = 1)
decrease	logical flag on how to sort, default = FALSE: increasing

Value

a character vector

Examples

```
elementsInFormulas(c(O = 2, C = 1),
                   c(H = 2, S = 1))
```

---

elementsMonoisotopic *generates a pre-defined R6 elements object which contains info on elements, mass values are mono isotopic*

Description

generates a pre-defined R6 elements object which contains info on elements, mass values are mono isotopic

Usage

```
elementsMonoisotopic()
```

Value

an elements object

Note

this object is (by default) used in all chemical calculations in this package

electron is not really meant to be used in formulas, but is needed to calculate mass & m/z of ions

Examples

```
print(elementsMonoisotopic())
```

`emptyFormula` *generates an empty pre-defined formula*

Description

generates an empty pre-defined formula

Usage

```
emptyFormula()
```

Value

a named numeric vector (formula)

Note

this can be used for calculations and setup of unknowns

Examples

```
print(emptyFormula())
```

`formulaString` *Translates regular formula format into a character vector, eg C6H12O6*

Description

Translates regular formula format into a character vector, eg C6H12O6

Usage

```
formulaString(formula, removeSingle = FALSE, useMarkdown = FALSE)
```

Arguments

`formula`	named numeric vector, example c(O = 2, C = 1)
`removeSingle`	if TRUE then for elements that are present in the formula only a single time, the number (1) will not be included. Default is FALSE. See also examples
`useMarkdown`	default = FALSE. If TRUE, the it will use HTML/Markdown codes <sub> in the formulas, which can be used with the library 'gt' to generate 'proper' notation for chemical formulas (numbers in subscript)

Value

character vector

Examples

```
formulaString(c(C=6, H=12, O=6))
formulaString(c(H=3,O=4,P=1))
formulaString(c(H=3,O=4,P=1), removeSingle = TRUE)
formulaString(c(H=2, O=1))
formulaString(c(H=2, O=1), removeSingle = TRUE)
```

`formulaToMass` *calculates the neutral mono-isotopic mass of a formula*

Description

calculates the neutral mono-isotopic mass of a formula

Usage

```
formulaToMass(
  formula = NULL,
  removeNA = FALSE,
  elementsInfo = elementsMonoisotopic(),
  enviPat = FALSE,
  exact = TRUE
)
```

Arguments

formula	named numeric vector, example c(O = 2, C = 1)
removeNA	logical vector: what to do if any of the elements is NA. If TRUE, then remove before calculation, if FALSE, then do not remove
elementsInfo	elements masses to be used, needs to be of class elements, default is elementsMonoisotopic(). The elementsAverage() function does not produce 100 complex isotope patterns that emerge. In case average masses are needed it's better to use the enviPat option
enviPat	logical argument that determines if the enviPat based calculations should be used. Default is FALSE. For monoisotopoc masses there is no difference, but for average masses of larger molecules (with complicated isotope patterns) it's highly recommended to use enviPat = TRUE with exact = FALSE
exact	determines if the exact (TRUE, default) or the average (FALSE) mass is calculated (ignored if enviPat is FALSE)#'

Value

numeric vector

Examples

```
formulaToMass(c(H=2, O=1))
formulaToMass(c(H=2, O=1), elementsInfo = elementsAverage())
formulaToMass(c(C = 50, H=102))
formulaToMass(c(C = 50, H=102), elementsInfo = elementsAverage())
formulaToMass(c(C = 50, H=102), enviPat = TRUE)
formulaToMass(c(C = 50, H=102), enviPat = TRUE, exact = FALSE)
```

---

massToMz *Calculates the m/z value of a charged/adducted ion*

Description

Calculates the m/z value of a charged/adducted ion

Usage

```
massToMz(
  mass,
  adducts = 0,
  adductFormula = electronFormula(),
  adductCharge = -1,
  elementsInfo = elementsMonoisotopic()
)
```

Arguments

mass	numeric vector, (neutral) mass of the molecule
adducts	numeric vector, number of adducts 'attached to' or 'removed from' the (originally neutral) molecule
adductFormula	formula (named numeric vector) of the adduct
adductCharge	numeric vector indicating the actual charge per adduct
elementsInfo	elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Value

numeric vector

Examples

```
# amino acid residue lysine + water
lysineMass <- formulaToMass(aminoAcidResidues()$getFormula("K") %f+% waterFormula())
lysineMass
# M+H+ : adducts =  1, adductFormula = protonFormula(), adductCharge = 1
# singly charged/protonated ion (ESI)
massToMz(lysineMass,
         adducts = 1,
         adductFormula = protonFormula(),
         adductCharge = 1)
# Doubly charged/protonated ion (ESI)
massToMz(lysineMass,
         adducts = 2,
         adductFormula = protonFormula(),
         adductCharge = 1)
# M-H- : single, negatively charged
massToMz(lysineMass,
         adducts = -1,
         adductFormula = protonFormula(),
         adductCharge = 1)
# M+ : singly positively charged (molecular) ion (EI)
massToMz(lysineMass,
         adducts = -1,
         adductFormula = electronFormula(),
         adductCharge = 1)
```

```
# M- : singly negatively charged (molecular) ion (EI)
massToMz(lysineMass,
         adducts = 1,
         adductFormula = electronFormula(),
         adductCharge = 1)
```

---

massToMzH *Calculates the m/z value of a protonated ion (positive ESI)*

Description

a wrapper around massToMz for positively charged, protonated ions in ESI

Usage

```
massToMzH(mass, charge = 1, elementsInfo = elementsMonoisotopic())
```

Arguments

mass	numeric vector, (neutral) mass of the molecule
charge	charge state
elementsInfo	elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Value

numeric vector

Examples

```
# amino acid residue lysine + water
lysineMass <- formulaToMass(aminoAcidResidues()$getFormula("K") %f+% waterFormula())
lysineMass
# M+H+ : adducts =  1, adductFormula = protonFormula(), adductCharge = 1
# singly charged/protonated ion (ESI)
massToMz(lysineMass,
         adducts = 1,
         adductFormula = protonFormula(),
         adductCharge = 1)
massToMzH(lysineMass)
```

`modifications` *R6 Class representing a set of modifications for the aminoacids in peptides*

Description

Every modification inside the object has a name, position, fixed (flag), gain (formula), loss (formula) and category.

'name' is a character vector.

Position is a character vector specifying the amino acids which always have the modification (fixed = TRUE) or can have the modification (fixed = FALSE). More than one amino acid can be specified, eg NQ (for Asparagine & glutamine). Please note that currently the package does NOT support 'exotic' amino acids (eg Selenocysteine) or 'combination' letters, such as 'J' (Leucine or Isoleucine). For the C- and N-terminus, use 'C_Term' or 'N_Term' for position.

Gain and loss specify what is lost and/or gained when a amino acid is modified. For example Carbamidomethylation of Cysteine has both a loss formula c(H=1) and a gain formula c(C=2, H=4, N=1, O=1); obviously this could also be defined as: loss formula = emptyFormula(), gain formula = c(C=2, H=3, N=1, O=1).

For the category field (character vector) there is no real 'rule' on how to classify modifications. I usually stick to the categorisation of Mascot or Sequest.

Active bindings

`number` retrieve the number of modifications present in the object, read only

`fixed` retrieve a table of the fixed modifications in the modification table, read only

`variable` retrieve a table of the variable modifications in the modification table, read only

`table` to access the table of modifications directly

Methods

Public methods:

- `modifications$new()`
- `modifications$print()`
- `modifications$add()`
- `modifications$addTable()`
- `modifications$clone()`

Method `new()`**:** Create a new modifications object

Usage:

```
modifications$new(data = NA)
```

Arguments:

`data` default = NA. If not NA, then should be a tibble with 6 columns: name, position, fixed, gain, loss and category. This is checked, but the contents of each column are not checked.

Method `print()`**:** For printing purposes: prints a table of the modifications

Usage:

```
modifications$print(...)
```

Arguments:

`...` no arguments, the function takes care of printing

Method `add()`**:** Adds a single modification

Usage:

```
modifications$add(name, position, fixed, gain, loss, category)
```

Arguments:

`name` character vector

`position` character vector, should be a valid amino acid residue

`fixed` logical vector, specifies whether the modification is fixed (TRUE) or dynamic (FALSE)

`gain` named numeric vector (formula) that specifies what (atoms) are gained when a modification is applied to an amino acid

`loss` named numeric vector (formula) that specifies what (atoms) are lost when a modification is applied to an amino acid

`category` character vector. Not rigidly defined: for user to be able to select/filter etc which type of modifications to use

Returns: nothing

Method `addTable()`**:** Add (a set of) modifications via a tibble

Usage:

```
modifications$addTable(data = NA)
```

Arguments:

`data` default = NA. If not NA, then should be a tibble with 6 columns: name, position, fixed, gain, loss and category. This is checked, but the contents of each column are not checked.

Returns: nothing

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
modifications$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

the logical vector fixed is very important. If TRUE, then a modification is considered to be always present, if FALSE then its presence is optional.

Examples

```
aaModifications <- modifications$new()
aaModifications$addTable(
  tibble::tibble(
    name = c("Carbamidomethyl (C)",
             "Carboxymethyl (C)",
             "Oxidation (M)"),
    position = c("C","C","M"),
    fixed = c(TRUE,TRUE,FALSE),
    gain = list(c(C = 2, H = 4, N = 1, O = 1),
                c(C = 2, H = 3, N = 0, O = 2),
                c(C = 0, H = 0, N = 0, O = 1)),
    loss = list(c(protonFormula()),
                c(protonFormula()),
                c(emptyFormula())),
    category = c("Cys-state","Cys-state","Preparation Artefact")
  )
)
aaModifications
aaModifications$add(name = "Deamidation",
                    position = "NQ",
                    fixed = FALSE,
                    gain = c(O = 1, H = 1),
                    loss = c(N = 1, H =2),
                    category = "Preparation Artefact")
aaModifications
```

`mzHToMass` *calculates the mass of the molecule in an ion (M+xH)x+*

Description

a wrapper around `mzToMass` for positively charged, protonated ions in ESI

Usage

```
mzHToMass(mz, charge = 1, elementsInfo = elementsMonoisotopic())
```

Arguments

`mz`	numeric vector, mass to charge ratio of he ion
`charge`	charge state
`elementsInfo`	elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Value

nuemric vector

Examples

```
massToMzH(mass = 174.1117, charge = 2) |> mzHToMass(charge = 2)
massToMzH(mass = 174.1117, charge = 1) |> mzHToMass(charge = 1)
```

`mzToMass` *calculates the mass of the molecule in an ion*

Description

essentially the reverse of the `massToMz`

Usage

```
mzToMass(
  mz,
  adducts = 0,
  adductFormula = electronFormula(),
  adductCharge = -1,
  elementsInfo = elementsMonoisotopic()
)
```

Arguments

`mz`	mass to charge ratio of the ion
`adducts`	numeric vector, number of adducts 'attached to' or 'removed from' the (originally neutral) molecule
`adductFormula`	formula (named numeric vector) of the adduct
`adductCharge`	numeric vector indicating the actual charge per adduct
`elementsInfo`	elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Value

numeric vector

Examples

```
massToMz(mass = 174.1117, adductFormula = c(e=1), adducts = 2, adductCharge = -1) |>
 mzToMass(adductFormula = c(e=1), adducts = 2, adductCharge = -1)
massToMz(mass = 174.1117, adductFormula = c(H=1), adducts = 1, adductCharge = 1) |>
 mzToMass(adductFormula = c(H=1), adducts = 1, adductCharge = 1)
```

pdToFormula *translates a proteome Discoverer (Thermo Scientific) elements formula string to a formula as used by this package*

Description

translates a proteome Discoverer (Thermo Scientific) elements formula string to a formula as used by this package

Usage

```
pdToFormula(pdFormula)
```

Arguments

pdFormula	a character vector. Formula in a format as used by proteome discoverer software

Value

formula of format c(H=2, O=1)

Examples

```
glucose <- pdToFormula("C(6) H(12) O(6)")
glucose
water <- pdToFormula("H(2) O")
water
```

peptide *R6 Class representing a (single) peptide*

Description

Contains two character vectors: one representing the amino acid sequence, and a second conatining info on the positions of 'variable' modifications. The object also contains a modification table specifying the 'fixed' amd 'variable' modifications.

Active bindings

`sequence` returns the amino acid sequence as a character vector, can be set but is not checked against the length of the modifications string

`length` returns the length of the peptide (read only)

`modifications` returns the moficiations string, can be set but is not checked agains the length of the sequence string

`modificationsTable` returns the mofication table, can be modified. Note: 'variable' modifications should match the modifications string

Methods

Public methods:

- peptide$new()
- peptide$print()
- peptide$sequence.part()
- peptide$modifications.part()
- peptide$modifications.formula.part()
- peptide$modifications.formula()
- peptide$formula.part()
- peptide$formula()
- peptide$mass.part()
- peptide$mass()
- peptide$mz.part()
- peptide$mz()
- peptide$mzH.part()
- peptide$mzH()
- peptide$fragments.part()
- peptide$fragments()
- peptide$fragments.part.immoniumIons()
- peptide$fragments.immoniumIons()
- peptide$clone()

Method `new()`**:** Create a new peptide object

Usage:

```
peptide$new(sequence = "", modificationTable = NA, variableModifications = NA)
```

Arguments:

`sequence` character vector, the amino acid sequence of the peptide

`modificationTable` the table from a R6 'modifications' object containing the variable and fixed modifications present in the amino acid sequence

`variableModifications` character vector specifying the position of variable modifications. The length of this vector must be the same length as the sequence. Each character specifies the modification at that position, eg "00010", means that position 1,2,3 & 5 are unmodified, while position 4 has the third variable modification in the the modification table. Note that the numbering follows the original row order of the modification table (fixed modifications filtered out). Additions to a modification table should not be a problem, deletions or editing can cause problems however as the object currently cannot deal with this itself. If this character vector is NA, then a character vector of "0"'s will be created (with the same length as the sequence)

Returns: a new 'peptide' object

Method `print()`**:** For printing purposes: prints the sequence string, the variable modifications string and the modification table

Usage:

```
peptide$print(...)
```

Arguments:

`...` no arguments, the function takes care of printing

Method `sequence.part()`**:** Retrieve part of the amino acid squence. Note: intended for internal use

Usage:

```
peptide$sequence.part(startSeq = 1L, endSeq = 1L)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the amino acid sequence to retrieve

`endSeq` integer vector, specifies the end of the part of the amino acid sequence to retrieve

Returns: character vector

Method `modifications.part()`**:** Retrieve part of the variable modification string. Note: intended for internal use

Usage:

```
peptide$modifications.part(startSeq = 1L, endSeq = 1L)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the variable modification string to retrieve

`endSeq` integer vector, specifies the end of the part of the variable modification string to retrieve

Returns: character vector

Method `modifications.formula.part()`**:** Determines the gain & loss formulas for a part of the peptide (waviable modification string and modification table are used for this): adds up all the losses and gains. If the position of a variable modification in the variable modification string does not match the amino acid in the modification table, then a warning is produced

Usage:

```
peptide$modifications.formula.part(
  startSeq = 1L,
  endSeq = 1L,
  Nterminal = TRUE,
  Cterminal = TRUE
)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the variable modification string to retrieve

`endSeq` integer vector, specifies the end of the part of the variable modification string to retrieve

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

Returns: a list of 2 formulas: the summed up gain formulas & the summed up loss formulas which are present in the part selected by startSeq and endSeq)

Method `modifications.formula()`**:** Deterines the gain & loss formulas for the full length of the peptide sequence. Essentially a wrapper for modifications.formula.part

Usage:

```
peptide$modifications.formula(Nterminal = TRUE, Cterminal = TRUE)
```

Arguments:

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

Returns: a list of 2 formulas: the summed up gain formulas & the summed up loss formulas which are present in the part selected by startSeq and endSeq)

Method `formula.part()`**:** Determines the chemical formula of part of the peptide with or without the modifications.

Usage:

```
peptide$formula.part(
  startSeq = 1,
  endSeq = 1,
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE
)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the peptide sequence

`endSeq` integer vector, specifies the end of the part of the peptide sequence

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

Returns: a named numeric vector, eg: c(C=6, H=12, O=6)

Method `formula()`**:** Determines the chemical formula of the full length peptide with or without modifications. Essentially a wrapper around 'formula.part'

Usage:

```
peptide$formula(
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE
)
```

Arguments:

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

Returns: a named numeric vector, eg: c(C=6, H=12, O=6)

Method `mass.part()`**:** Calculate the mass of part of the peptide with or without modifications

Usage:

```
peptide$mass.part(
  startSeq = 1,
  endSeq = 1,
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE,
  elementsInfo = elementsMonoisotopic()
)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the peptide sequence

`endSeq` integer vector, specifies the end of the part of the peptide sequence

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`elementsInfo` elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Returns: numeric vector

Method `mass()`**:** Calculate the mass of the full length peptide with or without modifications

Usage:

```
peptide$mass(
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE,
  elementsInfo = elementsMonoisotopic()
)
```

Arguments:

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`elementsInfo` elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Returns: numeric vector

Method `mz.part()`**:** Calculate the m/z of part of the peptide (as an ion) with or without modifications

Usage:

```
peptide$mz.part(
  startSeq = 1,
  endSeq = 1,
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE,
  elementsInfo = elementsMonoisotopic(),
  adducts = 1,
  adductFormula = protonFormula(),
  adductCharge = 1
)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the peptide sequence

`endSeq` integer vector, specifies the end of the part of the peptide sequence

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`elementsInfo` elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

`adducts` numeric vector, number of adducts attached to' or 'removed from' the (originally neutral) peptide

`adductFormula` formula (named numeric vector) of the adduct

`adductCharge` numeric vector indicating the actual charge per adduct

Returns: numeric vector

Method `mz()`**:** Calculate the m/z of the full length peptide (as an ion) with or without modifications

Usage:

```
peptide$mz(
  ignoreModifications = FALSE,
```

```
  Nterminal = TRUE,
  Cterminal = TRUE,
  elementsInfo = elementsMonoisotopic(),
  adducts = 1,
  adductFormula = protonFormula(),
  adductCharge = 1
)
```

Arguments:

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`elementsInfo` elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

`adducts` numeric vector, number of adducts attached to' or 'removed from' the (originally neutral) peptide

`adductFormula` formula (named numeric vector) of the adduct

`adductCharge` numeric vector indicating the actual charge per adduct

Returns: numeric vector

Method `mzH.part()`**:** Calculate the m/z of part of the peptide (as a protonated ion) with or without modifications

Usage:

```
peptide$mzH.part(
  startSeq = 1,
  endSeq = 1,
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE,
  charge = 1,
  elementsInfo = elementsMonoisotopic()
)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the peptide sequence

`endSeq` integer vector, specifies the end of the part of the peptide sequence

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`charge` charge state

`elementsInfo` elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Returns: numeric vector

Method `mzH()`**:** Calculate the m/z of part of the peptide (as a protonated ion) with or without modifications

Usage:

```
peptide$mzH(
  charge = 1,
  ignoreModifications = FALSE,
  Nterminal = TRUE,
  Cterminal = TRUE,
  elementsInfo = elementsMonoisotopic()
)
```

Arguments:

`charge` charge state

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide. Note: if TRUE then the 'Nterminal' and 'Cterminal' parameters are ignored

`Nterminal` logical vector if TRUE then Nterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`Cterminal` logical vector if TRUE then Cterminal modifications are included (if N-terminus is present in the part selected by startSeq and endSeq)

`elementsInfo` elements masses to be used, needs to be of class elements, default is elementsMonoisotopic()

Returns: numeric vector

Method `fragments.part()`**:** generates a table of fragments which could arise from fragmenting part of the peptide. The ionseries generated are: a, a-H2O, a-NH3, b, b-H2O, b-NH3, b+H2O, c, x, y, y-H2O, y-NH3, z. Please note that the calculation is relatively 'dumb': it does NOT check whether a fragment is possible at all. Prime example is the B+H2O ion series: these fragment ions can only if certain conditions are met. Currently there is no check in this function that checks these conditions/assumptions

Usage:

```
peptide$fragments.part(
  startSeq = 1,
  endSeq = 1,
  ignoreModifications = FALSE,
  onlyIons = TRUE,
  chargeState = 1,
  returnFormulas = FALSE,
  formulaIncludeChargeProtons = FALSE
)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the peptide sequence

`endSeq` integer vector, specifies the end of the part of the peptide sequence

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide

`onlyIons` default = TRUE, only information on the 13 (earlier mentioned) ion series is generated. If FALSE then an additional 10 columns are generated with info on the ionseries

`chargeState` charge state of the ions in the generated table

`returnFormulas` default = FALSE, if TRUE then in stead of numerical values the table will be populated by the chemical formulas of the neutral fragments or charged fragment ions

`formulaIncludeChargeProtons` default = FALSE, if TRUE then protons will be included in the formulas (ignored when ' returnFormulas = FALSE)

Returns: a data.frame with fragment information

Method `fragments()`**:** generates a table of fragments which could arise from fragmenting the full sequence of the peptide. The ion series generated are: a, a-H2O, a-NH3, b, b-H2O, b-NH3, b+H2O, c, x, y, y-H2O, y-NH3, z. Please note that the calculation is relatively 'dumb': it does NOT check whether a fragment is possible at all. Prime example is the B+H2O ion series: these fragment ions can only if certain conditions are met. Currently there is no check in this function that checks these conditions/assumptions

Usage:

```
peptide$fragments(
  ignoreModifications = FALSE,
  onlyIons = TRUE,
  chargeState = 1,
  returnFormulas = FALSE,
  formulaIncludeChargeProtons = FALSE
)
```

Arguments:

`ignoreModifications` if FALSE then modifications (both fixed & variable) are taken into account when calculating the chemical formula of the peptide

`onlyIons` default = TRUE, only information on the 13 (earlier mentioned) ion series is generated. If FALSE then an additional 10 columns are generated with info on the ionseries

`chargeState` charge state of the ions in the generated table

`returnFormulas` default = FALSE, if TRUE then in stead of numerical values the table will be populated by the chemical formulas of the neutral fragments or charged fragment ions

`formulaIncludeChargeProtons` default = FALSE, if TRUE then protons will be included in the formulas (ignored when ' returnFormulas = FALSE)

Returns: a data.frame with fragment information

Method `fragments.part.immoniumIons()`**:** generates a numeric vector containing 'expected' immonium ions based on the amino acid content of part of the peptide. Please note that this function does NOT take into account possible (fixed or variable) modifications

Usage:

```
peptide$fragments.part.immoniumIons(startSeq = 1, endSeq = 1)
```

Arguments:

`startSeq` integer vector, specifies the start of the part of the peptide sequence

`endSeq` integer vector, specifies the end of the part of the peptide sequence

Returns: numeric vector

Method `fragments.immoniumIons()`**:** generates a numeric vector containing 'expected' immonium ions based on the amino acid content of the full sequence of the peptide. Please note that this function does NOT take into account possible (fixed or variable) modifications

Usage:

```
peptide$fragments.immoniumIons()
```

Returns: numeric vector

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
peptide$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
testPeptide <- peptide$new(sequence = "SAMPLER",
                           modificationTable = aminoAcidModifications()$table,
                           variableModifications = "0010000")
testPeptide
testPeptide$formula()
```

`peptideCount` *counts the occurence of a amino acid (sequence) in another amino acid sequence*

Description

counts the occurence of a amino acid (sequence) in another amino acid sequence

Usage

```
peptideCount(
  thePeptide = NA,
  searchPeptide = NA,
  doNotSplice = TRUE,
  upper = TRUE
)
```

Arguments

thePeptide	character vector, the peptide to be searched
searchPeptide	character vector, the amino acid sequence to search for
doNotSplice	if FALSE the all characters in the searchPeptide are searched individually.If TRUE then the searchPeptide is searched as a whole. Default = TRUE
upper	convert both thePeptide & searchPeptides to uppercase before searching

Value

numeric vector

Examples

```
peptideCount("SAMPLER", "P")
peptideCount("SAMPLER", "PLER", doNotSplice = TRUE)
peptideCount("SAMPLER", "PLER", doNotSplice = FALSE)
```

---

peptideFormula *peptideFormula*

Description

gives formula of a peptide string

Usage

```
peptideFormula(peptide, aminoAcids = aminoAcidResidues())
```

Arguments

peptide	character vector specifying the sequence of amino acids in a peptide
aminoAcids	R6 object of type 'chemicals' with the amino acid info, default = aminoAcidResidues()

Value

a numeric vector

Note

does not check for non-amino acid letters, modifications cannot be specified

Examples

```
peptideFormula("SAMPLER")
```

peptideFragments — *Generates a pre-defined (incomplete) table of names of fragments for the ions resulting when fragmenting a peptide in MS*

Description

Generates a pre-defined (incomplete) table of names of fragments for the ions resulting when fragmenting a peptide in MS

Usage

```
peptideFragments()
```

Value

a data.frame of two columns: 'name' and 'series name' of fragments

Note

will possibly be removed in the future

peptideMzH — *peptideMzH*

Description

gives ion m/z of the protonated peptide

Usage

```
peptideMzH(
  peptide,
  charge = 1,
  aminoAcids = aminoAcidResidues(),
  elementsInfo = elementsMonoisotopic()
)
```

Arguments

peptide	character vector specifying the sequence of amino acids in a peptide
charge	numeric vector specifying the charge of the peptide ion
aminoAcids	R6 object of type 'chemicals' with the amino acid info, default = aminoAcidResidues()
elementsInfo	R6 object of type 'elements' with the elements masses info, default = elementsMonoisotopic()

Value

a numeric vector

Note

does not check for non-amino acid letters, modifications cannot be specified

Examples

```
peptideMzH("SAMPLER")
peptideMzH("SAMPLER", charge = 2)
peptideMzH("SAMPLER", elementsInfo = elementsAverage())
```

---

protonFormula *generates a pre-defined formula for proton*

Description

generates a pre-defined formula for proton

Usage

```
protonFormula()
```

Value

a named numeric vector (formula)

Examples

```
print(protonFormula())
```

---

rcdkFormula *translates an cdkFormula object to a 'regular' formula format*

Description

translates an cdkFormula object to a 'regular' formula format

Usage

```
rcdkFormula(cdkformula)
```

Arguments

cdkformula	an object of type rcdkFormula

Value

formula of format c(H=2, O=1)

Note

This function does not deal with the charge state which is possibly defined in the rcdkFormula object

Examples

```
glucose <- rcdk::get.formula("C6H12O6")
rcdkFormula(glucose)
glucoseAdductIon <- rcdk::get.formula("C6H12O6Na1", charge = 1)
glucoseAdductIon
# to get to the same m/z value
glucoseAdductIon |> massSpectrometryR::rcdkFormula() |> formulaToMass() |> massToMz(adducts = -1)
```

`removeZeros` *removeZeros*

Description

removes elements that have number zero

Usage

```
removeZeros(formula)
```

Arguments

`formula`	named numeric vector, example c(O = 2, C = 1)

Value

named numeric vector (formula)

Examples

```
glucose = c(O=6, H=12, C=6)
glucose %f+% emptyFormula()
removeZeros(glucose %f+% emptyFormula())
```

sortFormula *sortFormula*

Description

sorts the elements of a formula in alphabetical order (increasing/decreasing)

Usage

```
sortFormula(formula, decrease = FALSE)
```

Arguments

formula	named numeric vector, example c(O = 2, C = 1)
decrease	logical flag on how to sort, default = FALSE: increasing

Value

named numeric vector (formula)

Examples

```
glucose = c(O=6, H=12, C=6)
glucose
sortFormula(glucose)
```

stringFormula *Translates a character vector formula, eg 'C6H12O6' to a regular formula c(C=6, H=12, O=6)*

Description

Translates a character vector formula, eg 'C6H12O6' to a regular formula c(C=6, H=12, O=6)

Usage

```
stringFormula(string)
```

Arguments

string	character vector, format eg: 'C6H12O6'

Value

formula of format c(H=2, O=1)

Note

it's imperative that every element has a number (count), otherwise this function is highly likely to malfunction and return NA

Examples

```
stringFormula("H3O4P1")
stringFormula("C6H12O6")
```

`stringToFormula` *Translates a character vector formula, eg 'C6H12O6' to a regular formula c(C=6, H=12, O=6)*

Description

Translates a character vector formula, eg 'C6H12O6' to a regular formula c(C=6, H=12, O=6)

Usage

```
stringToFormula(string)
```

Arguments

`string`	character vector, format eg: 'C6H12O6'

Value

formula of format c(H=2, O=1)

Note

this function is an improved version of stringFormula(). Now every elements with count 1 can have the number omitted. However, the function depends on 'correct' elements (first letter is uppercase, second letter is lowercase). This function also allows for the presence of isotopes, eg '[13]C' or '[2]H2O'

Examples

```
stringToFormula("H3O4P1")
stringToFormula("C6H12O6")
stringToFormula("C6H5Br")
stringToFormula("[13]C6H12O5[18]O")
```

subtractFormulas *subtracting one formula from another, taking into account possible differing elements*

Description

subtracting one formula from another, taking into account possible differing elements

Usage

```
subtractFormulas(formula1, formula2)
```

Arguments

formula1	named numeric vector, example c(O = 2, C = 1); formula to be subtracted from
formula2	named numeric vector, example c(H = 2, S = 1); formula to subtract

Value

a named numeric vector (formula)

Note

There are no checks for negative values!

Examples

```
subtractFormulas(c(H = 2, O = 1), c(H = 1))
subtractFormulas(c(H = 2, O = 1), c(S = 1, O = 2))
```

validFormula *checks if formula is valid*

Description

checks if formula is valid

Usage

```
validFormula(formula, string = FALSE)
```

Arguments

formula	character vector or named numeric vector, representing the formula to be checked
string	logical vector specifying if the formula is a character vector or not

Value

logical vector

Note

This is done via the check_chemform function from the package enviPat

Examples

```
glucose <- c(C=6, H=12, O=6)
validFormula(glucose)
formulaString(glucose)
validFormula(formulaString(glucose), string = TRUE)
```

`waterFormula` *generates a pre-defined formula for water*

Description

generates a pre-defined formula for water

Usage

```
waterFormula()
```

Value

a named numeric vector (formula)

Examples

```
print(waterFormula())
```

`%f-%` *custom operator for subtracting formulas from one another, to make calculating with formulas a little more clear*

Description

custom operator for subtracting formulas from one another, to make calculating with formulas a little more clear

Usage

```
formula1 %f-% formula2
```

Arguments

formula1	named numeric vector, example c(O = 2, C = 1); formula to be subtracted from
formula2	named numeric vector, example c(H = 2, S = 1); formula to subtract

Examples

```
c(H = 2, O = 1) %f-% c(H = 1)
c(H = 2, O = 1) %f-% c(S = 1, O = 2)
```

%f+% *custom operator for adding up formulas, to make calculating with formulas a little more clear*

Description

custom operator for adding up formulas, to make calculating with formulas a little more clear

Usage

```
formula1 %f+% formula2
```

Arguments

formula1	named numeric vector, example c(O = 2, C = 1)
formula2	named numeric vector, example c(H = 2, S = 1)

Examples

```
waterFormula() %f+% protonFormula()
waterFormula() %f+% c(C=1, O = 2)
c(H = 2, O = 1) %f+% c(S = 1, O = 2)
```

Index