Package 'debkeepr'

July 22, 2025

Type Package

Title Analysis of Non-Decimal Currencies and Double-Entry Bookkeeping

Version 0.1.1

Description Analysis of historical non-decimal currencies and value systems that use tripartite or tetrapartite systems such as pounds, shillings, and pence. It introduces new vector classes to represent non-decimal currencies, making them compatible with numeric classes, and provides functions to work with these classes in data frames in the context of double-entry bookkeeping.

License MIT + file LICENSE

URL https://github.com/jessesadler/debkeepr, https://jessesadler.github.io/debkeepr/

BugReports https://github.com/jessesadler/debkeepr/issues

Depends R (>= 3.5)

Imports cli (>= 3.4.0), dplyr (>= 1.0.0), magrittr, methods, rlang (>= 1.1.0), tibble (>= 3.0.0), vctrs (>= 0.5.2), zeallot

Suggests covr, ggplot2, ggraph, igraph, knitr, rmarkdown, roxygen2, scales (>= 1.1.0), testthat (>= 3.1.3)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Jesse Sadler [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-6081-9681>)

Maintainer Jesse Sadler <jrsadler@icloud.com>

Repository CRAN

Date/Publication 2023-03-22 09:00:10 UTC

Contents

`arithmetic` *Arithmetic operations for debkeepr*

Description

Arithmetic operations for debkeepr

Usage

```
## S3 method for class 'deb_lsd'
vec_arith(op, x, y, ...)

## Default S3 method:
vec_arith.deb_lsd(op, x, y, ...)

## S3 method for class 'deb_lsd'
vec_arith.deb_lsd(op, x, y, ...)

## S3 method for class 'numeric'
```

```
vec_arith.deb_lsd(op, x, y, ...)

## S3 method for class 'deb_lsd'
vec_arith.numeric(op, x, y, ...)

## S3 method for class 'MISSING'
vec_arith.deb_lsd(op, x, y, ...)

## S3 method for class 'deb_decimal'
vec_arith(op, x, y, ...)

## Default S3 method:
vec_arith.deb_decimal(op, x, y, ...)

## S3 method for class 'deb_decimal'
vec_arith.deb_decimal(op, x, y, ...)

## S3 method for class 'numeric'
vec_arith.deb_decimal(op, x, y, ...)

## S3 method for class 'deb_decimal'
vec_arith.numeric(op, x, y, ...)

## S3 method for class 'MISSING'
vec_arith.deb_decimal(op, x, y, ...)

## S3 method for class 'deb_tetra'
vec_arith(op, x, y, ...)

## Default S3 method:
vec_arith.deb_tetra(op, x, y, ...)

## S3 method for class 'deb_tetra'
vec_arith.deb_tetra(op, x, y, ...)

## S3 method for class 'numeric'
vec_arith.deb_tetra(op, x, y, ...)

## S3 method for class 'deb_tetra'
vec_arith.numeric(op, x, y, ...)

## S3 method for class 'MISSING'
vec_arith.deb_tetra(op, x, y, ...)

## S3 method for class 'deb_decimal'
vec_arith.deb_lsd(op, x, y, ...)

## S3 method for class 'deb_lsd'
```

```
vec_arith.deb_decimal(op, x, y, ...)

## S3 method for class 'deb_tetra'
vec_arith.deb_lsd(op, x, y, ...)

## S3 method for class 'deb_lsd'
vec_arith.deb_tetra(op, x, y, ...)

## S3 method for class 'deb_decimal'
vec_arith.deb_tetra(op, x, y, ...)

## S3 method for class 'deb_tetra'
vec_arith.deb_decimal(op, x, y, ...)
```

Arguments

op	Arithmetic operation.
x, y	Vectors.
...	For future expansion

Value

A deb_lsd, deb_tetra, deb_decimal or numeric vector depending on the inputs and arithmetic operator.

---

cast-decimal *Cast to* deb_decimal

Description

Cast x to a deb_decimal vector.

Usage

```
deb_as_decimal(x, ...)

## Default S3 method:
deb_as_decimal(x, ...)

## S3 method for class 'deb_decimal'
deb_as_decimal(x, ...)

## S3 method for class 'deb_lsd'
deb_as_decimal(x, unit = c("l", "s", "d"), ...)

## S3 method for class 'deb_tetra'
deb_as_decimal(x, unit = c("l", "s", "d", "f"), ...)
```

```
## S3 method for class 'numeric'
deb_as_decimal(x, unit = c("l", "s", "d", "f"), bases = c(20, 12), ...)

## S3 method for class 'logical'
deb_as_decimal(x, unit = c("l", "s", "d", "f"), bases = c(20, 12), ...)

## S3 method for class 'list'
deb_as_decimal(x, unit = c("l", "s", "d", "f"), bases = c(20, 12), ...)
```

Arguments

x	An object to coerce to deb_decimal.
...	Arguments passed on to further methods.
unit	A character vector of length one indicating the unit for the decimalized values, either "l" (libra, the default), "s" (solidus), or "d" (denarius).
bases	Numeric vector of length 2 used to specify the bases for the solidus or s and denarius or d units. Default is c(20, 12), which conforms to the most widely used system of 1 pound = 20 shillings and 1 shilling = 12 pence.

Details

Like deb_as_lsd(), deb_as_decimal() provides a method to cast a list of numeric vectors of length 3 to deb_decimal. This may be helpful because the data is input by the value instead of by the unit.

Value

A deb_decimal vector.

See Also

deb_as_lsd() and deb_as_tetra()

Examples

```
# Cast a deb_lsd vector to deb_decimal
x <- deb_lsd(l = c(5, 3, 7),
             s = c(16, 5, 6),
             d = c(6, 0, 8))
deb_as_decimal(x)

# Bases are automatically applied when
# casting from deb_lsd to deb_decimal
x2 <- deb_lsd(l = c(5, 3, 7),
              s = c(16, 5, 6),
              d = c(6, 0, 8),
              bases = c(60, 16))
deb_as_decimal(x2)
```

```
# Cast a deb_tetra vector to deb_decimal
# Bases are automatically applied, creating
# a deb_decimal vector with three bases units.
y <- deb_tetra(l = c(5, 13, 7),
               s = c(12, 8, 16),
               d = c(3, 11, 0),
               f = c(1, 3, 2))
deb_as_decimal(y)

# Cast a numeric vector to deb_decimal
z <- c(5.825, 3.25, 22/3)
deb_as_decimal(z)

# Use the unit and bases arguments to specify
# the unit and apply non-default bases
deb_as_decimal(z, unit = "s", bases = c(60, 16))

# Casting a list to deb_decimal provides an
# alternative to get lsd values to deb_decimal.
lsd_list <- list(c(5, 12, 3),
                 c(13, 8, 11),
                 c(7, 16, 0))
deb_as_decimal(lsd_list)
```

---

cast-lsd *Cast to* deb_lsd

Description

Cast x to a deb_lsd vector.

Usage

```
deb_as_lsd(x, ...)

## Default S3 method:
deb_as_lsd(x, ...)

## S3 method for class 'deb_lsd'
deb_as_lsd(x, ...)

## S3 method for class 'deb_decimal'
deb_as_lsd(x, ...)

## S3 method for class 'deb_tetra'
deb_as_lsd(x, ...)

## S3 method for class 'numeric'
```

```
deb_as_lsd(x, bases = c(20, 12), ...)

## S3 method for class 'logical'
deb_as_lsd(x, bases = c(20, 12), ...)

## S3 method for class 'list'
deb_as_lsd(x, bases = c(20, 12), ...)
```

Arguments

x	An object to coerce to deb_lsd.
...	Arguments passed on to further methods.
bases	Numeric vector of length 2 used to specify the bases for the solidus or s and denarius or d units. Default is c(20, 12), which conforms to the most widely used system of 1 pound = 20 shillings and 1 shilling = 12 pence.

Details

Casting a list of numeric vectors of length 3 to deb_lsd provides an alternate way to create a deb_lsd vector than deb_lsd(). This method may be helpful because the data is input by the value instead of by the unit.

Value

A deb_lsd vector.

See Also

deb_as_decimal() and deb_as_tetra()

Examples

```
# Cast a deb_decimal vector to deb_lsd
x <- c(5.825, 3.25, 22/3)
d1 <- deb_decimal(x)
deb_as_lsd(d1)

# Bases are automatically applied when
# casting from deb_decimal to deb_lsd
d2 <- deb_decimal(x, bases = c(60, 16))
deb_as_lsd(d2)

# Cast a deb_tetra vector to deb_lsd
# This removes the 'f' or farthings unit.
y <- deb_tetra(l = c(5, 13, 7),
               s = c(12, 8, 16),
               d = c(3, 11, 0),
               f = c(1, 3, 2))
deb_as_lsd(y)

# Cast a numeric vector to deb_lsd
```

```
deb_as_lsd(x)

# Use the bases argument to apply non-default bases
deb_as_lsd(x, bases = c(60, 16))

# Casting a list to deb_lsd provides an alternate to deb_lsd()
# This can be helpful for legibility. Compare:

deb_as_lsd(
  list(c(5, 12, 3),
       c(13, 8, 11),
       c(7, 16, 0))
  )

deb_lsd(l = c(5, 13, 7),
        s = c(12, 8, 16),
        d = c(3, 11, 0))
```

---

cast-tetra *Cast to* deb_tetra

Description

Cast x to a deb_tetra vector.

Usage

```
deb_as_tetra(x, ...)

## Default S3 method:
deb_as_tetra(x, ...)

## S3 method for class 'deb_tetra'
deb_as_tetra(x, ...)

## S3 method for class 'deb_lsd'
deb_as_tetra(x, f, ...)

## S3 method for class 'deb_decimal'
deb_as_tetra(x, f, ...)

## S3 method for class 'numeric'
deb_as_tetra(x, bases = c(20, 12, 4), ...)

## S3 method for class 'logical'
deb_as_tetra(x, bases = c(20, 12, 4), ...)

## S3 method for class 'list'
deb_as_tetra(x, bases = c(20, 12, 4), ...)
```

Arguments

x	An object to coerce to deb_tetra.
...	Arguments passed on to further methods.
f	Integer of length 1 to represent the base of the farthing unit. Must be provided to cast from deb_lsd or deb_decimal vectors with tripartite bases to deb_tetra.
bases	Numeric vector of length 3 used to specify the bases for the solidus or s, denarius or d, and farthing or f units. Default is c(20, 12, 4), which conforms to the English system of 1 pound = 20 shillings, 1 shilling = 12 pence, and 1 pence = 4 farthing.

Details

Casting a list of numeric vectors of length 4 to deb_tetra provides an alternate way to create a deb_tetra vector than deb_tetra(). This method may be helpful because the data is input by the value instead of by the unit.

Value

A deb_tetra vector.

See Also

deb_as_lsd() and deb_as_decimal()

Examples

```
# To cast from deb_lsd to deb_tetra an "f" unit must be supplied

# Compare
lsd1 <- deb_lsd(8, 12, 4)
lsd2 <- deb_lsd(8, 12, 4, bases = c(60, 16))

deb_as_tetra(lsd1, f = 4)
deb_as_tetra(lsd2, f = 8)

# Cast a deb_decimal vector with four units to deb_tetra.
# Bases are automatically applied when casting from
# tetrapartite deb_decimal to deb_tetra.
x <- c(5.11875, 3.76875, 25/3)
d1 <- deb_decimal(x, bases = c(20, 12, 4))
deb_as_tetra(d1)

# Use "f" argument to cast from tripartite deb_decimal
# to deb_tetra
d2 <- deb_decimal(x)
deb_as_tetra(d2, f = 4)

# Cast a numeric vector to deb_tetra
deb_as_tetra(x)
```

```
# Use the bases argument to apply non-default bases
deb_as_tetra(x, bases = c(60, 16, 8))

# Casting a list to deb_tetra provides an alternate
# to deb_tetra(). This can be helpful for legibility.
# Compare:

deb_as_tetra(
  list(c(5, 12, 3, 2),
       c(13, 8, 11, 1),
       c(7, 16, 0, 3))
  )

deb_tetra(l = c(5, 13, 7),
          s = c(12, 8, 16),
          d = c(3, 11, 0),
          f = c(2, 1, 3))
```

`comparison` *Equality and comparison*

Description

Equality and comparison

Usage

```
## S3 method for class 'deb_lsd'
vec_proxy_equal(x, ...)

## S3 method for class 'deb_lsd'
vec_proxy_compare(x, ...)

## S3 method for class 'deb_tetra'
vec_proxy_equal(x, ...)

## S3 method for class 'deb_tetra'
vec_proxy_compare(x, ...)
```

Arguments

`x`	A deb_lsd vector.
`...`	Arguments passed on to further methods.

Value

A data frame or numeric vector to be used for comparison.

convert-bases *Convert bases of* deb_lsd, deb_tetra, *and* deb_decimal *vectors*

Description

Convert bases of deb_lsd, deb_tetra, and deb_decimal vectors

Usage

```
deb_convert_bases(x, to)

## Default S3 method:
deb_convert_bases(x, to)

## S3 method for class 'deb_lsd'
deb_convert_bases(x, to)

## S3 method for class 'deb_decimal'
deb_convert_bases(x, to)

## S3 method for class 'deb_tetra'
deb_convert_bases(x, to)
```

Arguments

x	A vector of class deb_lsd, deb_tetra, or deb_decimal.
to	Numeric vector of length 2 or 3, representing the bases for the solidus, denarius, and optionally farthing units to be converted to.

Details

deb_convert_bases() is the only way to change the bases of the solidus, denarius, and farthing units associated with vectors of class deb_lsd, deb_tetra, and deb_decimal. It also provides a means to convert between tripartite and tetrapartite bases with deb_decimal vectors.

If x is a deb_decimal vector with tetrapartite bases and unit "f" and to is a numeric vector of length 2, the unit will be converted to "d".

Value

A vector of the same class as x with converted bases attribute.

Examples

```
lsd <- deb_lsd(5, 3, 8)
dec <- deb_decimal(8.825)
dec_tetra <- deb_decimal(1.840625, bases = c(20, 12, 4))
tetra <- deb_tetra(1, 16, 9, 3)
```

```
deb_convert_bases(lsd, to = c(60, 16))
deb_convert_bases(dec, to = c(60, 16))
deb_convert_bases(dec_tetra, c(60, 16, 8))
deb_convert_bases(tetra, to = c(60, 16, 8))

# Convert between tripartite and tetrapartite bases
deb_convert_bases(dec, to = c(60, 16, 8))
deb_convert_bases(dec_tetra, to = c(20, 12))
```

`dafforne_accounts` *Accounts from the practice journal and ledger of Richard Dafforne*

Description

A data set of the accounts from the first practice journal and ledger in Richard Dafforne's accounting manual from 1660 called *The Merchant's Mirrour*. By 1660 *The Merchant's Mirrour* was in its third edition, and its contents had been printed in the well-known merchant manual of Gerard Malynes, *Consuetudo Vel Lex Mercatoira* since the 1636 edition, making it one of the most popular bookkeeping manuals in 17th-century England. The data set is meant to be used in conjunction with `dafforne_transactions`. It contains information on the accounts found in the practice journal and ledger that Dafforne used to teach double-entry bookkeeping practices.

Usage

```
dafforne_accounts
```

Format

A data frame with 46 rows and 5 variables.

Details

The data set does not include the Balance account that Dafforne uses to close the books. The transactions from this account can be recreated using the lsd account functions in `debkeepr`.

Variables

- `id`: Numeric id for each account. The ids correspond to the ids in the `credit` and `debit` variables in `dafforne_transactions`.
- `account`: Name of the account.
- `ledger`: Page on which the account appears in the ledger.
- `investor`: The investor or the person's whose capital is involved in the account. Accounts that only deal with the bookkeeper's capital are listed as "ego".
- `description`: Short description of each account.

Source

Richard Dafforne, *The Merchant's Mirrour, Or Directions for the Perfect Ordering and Keeping of His Accounts*, Third Edition, (London, 1660)

`dafforne_transactions` *Transactions from the practice journal and ledger of Richard Dafforne*

Description

A data set of the transactions from the first practice journal and ledger in Richard Dafforne's accounting manual from 1660 called *The Merchant's Mirrour*. By 1660 *The Merchant's Mirrour* was in its third edition, and its contents had been printed in the well-known merchant manual of Gerard Malynes, *Consuetudo Vel Lex Mercatoira* since the 1636 edition, making it one of the most popular bookkeeping manuals in 17th-century England. The data set is meant to be used in conjunction with `dafforne_accounts`. It contains the transactions in the practice journal and ledger that Dafforne used to teach double-entry bookkeeping practices.

Usage

```
dafforne_transactions
```

Format

A data frame with 177 rows and 8 variables.

Details

The data set does not include the last 16 transactions recorded in the journal, which deal with the balancing of the book. These transactions can be recreated using the lsd account functions in `debkeepr`.

Variables

- `id`: Numeric id for each transaction.
- `credit`: Account id for the credit account in the transactions. The accounts that discharges the transactional value or from which the value derives. The account ids correspond to the `id` variable in `dafforne_accounts`.
- `debit`: Account id for the debit account in the transactions. The accounts that receive the transactional value. The account ids correspond to the `id` variable in `dafforne_accounts`.
- `date`: Date on which the transaction was entered into the journal. Date conforms to the Anglican calendar that used the old Julian calendar with the new year on 25 March. Encoded as a date vector.
- `lsd`: Column of class `deb_lsd` with pounds, shillings, and pence values. Bases for shillings and pence are 20 and 12 respectively.
- `journal`: Page on which the transaction is recorded in the journal.

- `ledger`: The pages on which the transaction is recorded in the ledger. The number before the slash is the page on which the debit is recorded. The number after the slash is the page on which the credit is recorded.
- `description`: Description of the transaction as recorded in the journal.

Source

Richard Dafforne, *The Merchant's Mirrour, Or Directions for the Perfect Ordering and Keeping of His Accounts*, Third Edition, (London, 1660)

`deb_convert_unit` *Convert the unit of* `deb_decimal` *vectors*

Description

Convert the `unit` attribute of `deb_decimal` vectors.

Usage

```
deb_convert_unit(x, to = c("l", "s", "d", "f"))
```

Arguments

`x`	A vector of class `deb_decimal`.
`to`	A character vector of length one indicating the unit to be converted to. Choice of `"l"` (libra, the default), `"s"` (solidus), `"d"` (denarius), or `"f"` (farthing).

Details

`deb_convert_unit()` converts the `unit` of a `deb_decimal` vector to either `"l"`, `"s"`, `"d"`, or optionally `"f"` if the vector has tetrapartite bases. This changes the representation of the vector, but the value remains equivalent.

Value

A `deb_decimal` vector with a converted `unit` attribute.

Examples

```
x <- deb_decimal(c(8.825, 15.125, 3.65))
y <- deb_decimal(c(56.45, 106.525, 200.4), unit = "s")
z <- deb_decimal(c(8472, 14520,  3504),
                 unit = "f",
                 bases = c(20, 12, 4))

deb_convert_unit(x, to = "s")
deb_convert_unit(x, to = "d")
deb_convert_unit(y, to = "l")
```

```
deb_convert_unit(y, to = "d")
deb_convert_unit(z, to = "l")
deb_convert_unit(z, to = "s")
```

`deb_decimal` *A decimalized class for tripartite and tetrapartite values*

Description

Create a vector of class `deb_decimal` to integrate non-decimal currencies and other measurements that use tripartite or tetrapartite units into standardized forms of analysis provided by R.

Usage

```
deb_decimal(x = double(), unit = c("l", "s", "d", "f"), bases = c(20, 12))
```

Arguments

`x`	A numeric vector representing the decimalized values of either tripartite or tetrapartite values.
`unit`	A character vector of length one indicating the unit for the decimalized values, either `"l"` (*libra*, the default), `"s"` (*solidus*), `"d"` (*denarius*), or `"f"` (farthing). `"f"` is only valid if the `bases` argument is a numeric vector of length 3 (a tetrapartite value).
`bases`	Numeric vector of length 2 or 3 used to specify the bases for the solidus or s, denarius or d, and optionally the farthing or f units. Default is `c(20, 12)`, which conforms to the most widely used tripartite system of 1 pound = 20 shillings and 1 shilling = 12 pence.

Details

The `deb_decimal` class and the `debkeepr` package use the nomenclature of l, s, and d to represent the tripartite system of pounds, shillings, and pence units. The abbreviations derive from the Latin terms *libra*, *solidus*, and *denarius*. In the 8th century a *solidus* came to represent 12 *denarii* coins, and, for a time at least, 240 *denarii* were made from one *libra* or pound of silver. The custom of counting coins in dozens (*solidi*) and scores of dozens (*librae*) spread throughout the Carolingian Empire and became ingrained in much of Europe. However, a variety of accounting systems arose at different times that used other bases for the *solidus* and *denarius* units and even additional units. The `deb_decimal` class decimalizes either tripartite or tetrapartite values. The `bases` attribute makes it possible to specify the bases for the *solidus*, *denarius*, and optionally farthing units. The `unit` attribute identifies the decimalized unit: either *libra*, *solidus*, *denarius*, or farthing.

`deb_decimal` vectors can either be tripartite, like `deb_lsd`, or tetrapartite, like `deb_tetra`. These two kinds of `deb_decimal` vectors are distinguished by the length of `bases` attribute (2 for tripartite and 3 for tetrapartite) and the addition of the farthing unit for tetrapartite. If the *solidus* and *denarius* bases are equal, tripartite and tetrapartite `deb_decimal` vectors can be combined. The result is a `deb_decimal` vector with tripartite bases.

Value

Returns a vector of class deb_decimal.

See Also

The deb_decimal class works in concert with the deb_lsd and deb_tetra classes. These classes maintain the tripartite (deb_lsd) and tetrapartite (deb_tetra) unit structure of non-decimal currencies and values. See deb_lsd() and deb_tetra().

Examples

```
# deb_decimal with tripartite units
deb_decimal(c(5.25, 3.825, 8.5))

# Set the unit of the deb_decimal vector
deb_decimal(c(105, 76.5, 170), unit = "s")
deb_decimal(c(1260, 918, 240), unit = "d")

# Set the bases of the deb_decimal vector
deb_decimal(c(5.25, 3.825, 8.5), bases = c(60, 16))

# Create a prototype or vector of length 0
deb_decimal()

# To create a tetrapartite value, provide numeric vector
# of length 3 to bases argument
deb_decimal(c(5.11875, 3.234375, 8.2875),
            bases = c(20, 12, 4))
deb_decimal(c(4914, 3105, 7956),
            unit = "f",
            bases = c(20, 12, 4))
```

---

deb_is_decimal *Test if an object is of class* deb_decimal

Description

Test if an object is of class deb_decimal.

Usage

```
deb_is_decimal(x)
```

Arguments

x	An object.

Value

TRUE if object is of class deb_decimal and FALSE if it is not.

Examples

```
x <- deb_decimal(c(5.25, 3.825, 8.5))
y <- c(5.25, 3.825, 8.5)

deb_is_decimal(x)
deb_is_decimal(y)
```

---

deb_is_lsd *Test if an object is of class* deb_lsd

Description

Test if an object is of class deb_lsd.

Usage

```
deb_is_lsd(x)
```

Arguments

x	An object.

Value

TRUE if object is of class deb_lsd and FALSE if it is not.

Examples

```
x <- deb_lsd(5, 3, 8)
y <- c(5, 3, 8)

deb_is_lsd(x)
deb_is_lsd(y)
```

---

deb_is_tetra *Test if an object is of class* deb_tetra

Description

Test if an object is of class deb_tetra.

Usage

```
deb_is_tetra(x)
```

Arguments

x	An object.

Value

TRUE if object is of class deb_tetra and FALSE if it is not.

Examples

```
x <- deb_tetra(5, 3, 8, 2)
y <- c(5, 3, 8, 2)

deb_is_tetra(x)
deb_is_tetra(y)
```

---

deb_lsd *A class for pounds, shillings and pence values*

Description

Create a vector of class deb_lsd to integrate non-decimal currencies into standardized forms of analysis provided by R.

Usage

```
deb_lsd(l = double(), s = double(), d = double(), bases = c(20, 12))
```

Arguments

l	Numeric vector representing the pounds unit.
s	Numeric vector representing the shillings unit.
d	Numeric vector representing the pence unit.
bases	Numeric vector of length 2 used to specify the bases for the solidus or s and denarius or d units. Default is c(20, 12), which conforms to the most widely used system of 1 pound = 20 shillings and 1 shilling = 12 pence.

Details

The deb_decimal class and the debkeepr package use the nomenclature of l, s, and d to represent the tripartite system of pounds, shillings, and pence units. The abbreviations derive from the Latin terms *libra*, *solidus*, and *denarius*. In the 8th century a *solidus* came to represent 12 *denarii* coins, and, for a time at least, 240 *denarii* were made from one *libra* or pound of silver. The custom of counting coins in dozens (*solidi*) and scores of dozens (*librae*) spread throughout the Carolingian Empire and became ingrained in much of Europe. However, a variety of accounting systems arose at different times that used other bases for the *solidus* and *denarius* units. The bases attribute of deb_decimal vectors makes it possible to specify alternative bases for the *solidus* and *denarius* units.

The length of l, s, and d must either be all equal, or a vector of length 1 can be recycled to the length of the other argument(s). See the vctrs package for further details on recycling vectors. In addition, l, s, and d must either all have no values, resulting in a vector of length 0, or all possess numeric vectors.

Value

Returns a vector of class deb_lsd.

See Also

The deb_lsd class works in concert with the deb_decimal class, which represents non-decimal currencies as decimalized values. See deb_decimal(). To represent values with tetrapartite units see deb_tetra().

Examples

```
deb_lsd(5, 3, 8)
deb_lsd(l = c(10, 8, 5),
        s = c(6, 13, 8),
        d = c(8, 4, 10))

# Recycle length 1 vector
deb_lsd(l = c(10, 8, 5),
        s = c(6, 13, 8),
        d = 0)

# Set the bases of the deb_lsd vector
deb_lsd(5, 3, 8, bases = c(60, 16))
deb_lsd(l = c(10, 28, 5),
        s = c(6, 33, 13),
        d = c(8, 42, 10),
        bases = c(60, 16))

# Create a prototype or vector of length 0
deb_lsd()
```

---

deb_tetra *A class for tetrapartite values*

Description

Create a vector of class deb_tetra to integrate values with four units into standardized forms of analysis provided by R.

Usage

```
deb_tetra(
  l = double(),
  s = double(),
  d = double(),
  f = double(),
  bases = c(20, 12, 4)
)
```

Arguments

l	Numeric vector representing the pounds unit.
s	Numeric vector representing the shillings unit.
d	Numeric vector representing the pence unit.
f	Numeric vector representing the farthing or fourth unit.
bases	Numeric vector of length 3 used to specify the bases for the solidus or s, denarius or d, and farthing or f units. Default is `c(20, 12, 4)`, which conforms to the English system of 1 pound = 20 shillings, 1 shilling = 12 pence, and 1 pence = 4 farthing.

Details

The `deb_tetra` class extends the concept of the `deb_lsd` class to incorporate currencies and other types of values that consist of four units. A variety of currencies and measurements of weights expanded beyond the conventional tripartite system of pounds, shillings, and pence to include a fourth unit. `deb_tetra` adds a fourth unit, named `f` for farthing, to the l, s, and d units used by `deb_lsd`. The `bases` attribute of `deb_tetra` vectors makes it possible to specify alternative bases for the *solidus*, *denarius*, and farthing units.

The length of `l`, `s`, `d`, and `f` must either be all equal, or a vector of length 1 can be recycled to the length of the other argument(s). See the vctrs package for further details on recycling vectors. In addition, `l`, `s`, `d`, and `f` must either all have no values, resulting in a vector of length 0, or all possess numeric vectors.

Value

Returns a vector of class `deb_tetra`.

See Also

The `deb_tetra` class works in concert with the `deb_decimal` class, which can represent tetrapartite values as decimalized values. See `deb_decimal()`. To represent values with tripartite units see `deb_lsd()`.

Examples

```
deb_tetra(5, 3, 8, 2)
deb_tetra(l = c(10, 8, 5),
          s = c(6, 13, 8),
```

```
          d = c(8, 4, 10),
          f = c(2, 3, 1))

# Recycle length 1 vector
deb_tetra(l = c(10, 8, 5),
          s = c(6, 13, 8),
          d = c(8, 4, 10),
          f = 2)

# Set the bases of the deb_tetra vector
deb_tetra(5, 3, 8, 2, bases = c(60, 16, 8))
deb_tetra(l = c(10, 28, 5),
          s = c(6, 33, 13),
          d = c(8, 12, 10),
          f = c(5, 3, 6),
          bases = c(60, 16, 8))

# Create a prototype or vector of length 0
deb_tetra()
```

`list-lsd` *Cast* `deb_lsd` *or* `deb_tetra` *to a list of values*

Description

Cast a `deb_lsd` or `deb_tetra` vector to a list of numeric vectors either three or four values per list item corresponding to lsd or tetra values.

Usage

```
deb_as_list(x, ...)

## Default S3 method:
deb_as_list(x, ...)

## S3 method for class 'deb_lsd'
deb_as_list(x, ...)

## S3 method for class 'deb_tetra'
deb_as_list(x, ...)
```

Arguments

`x`	A `deb_lsd` or `deb_tetra` vector to cast to a list of values.
`...`	Arguments passed on to further methods.

Details

deb_as_list() turns a deb_lsd or deb_tetra vector into a list of numeric vectors of length 3 or 4. It is the inverse of deb_as_lsd() and deb_as_tetra(). Compare to as.list(), which creates a list of deb_lsd or deb_tetra vectors or unclass(), which creates a list of length 3 or 4 with numeric vectors corresponding to the units.

Value

A list of numeric vectors of length 3 or 4, corresponding to lsd or tetra values.

See Also

deb_as_lsd() and deb_as_tetra() for the inverse of deb_as_list().

Examples

```
# deb_lsd vector
x <- deb_lsd(l = 0:3, s = 4:7, d = 8:11)

deb_as_list(x)

# deb_tetra vector

y <- deb_tetra(l = 0:3, s = 4:7, d = 8:11, f = 1:4)

deb_as_list(y)

# This is the inverse of `deb_as_lsd()` of a list of lsd values
z <- deb_as_list(x)

identical(x, deb_as_lsd(z))
```

---

lsd-column *Helpers to create and separate a* deb_lsd *column in a data frame*

Description

- deb_gather_lsd() creates a deb_lsd column from separate variables representing pounds, shillings, and pence values.
- deb_spread_lsd() creates separate variables for pounds, shillings, and pence from a deb_lsd column.

Usage

```
deb_gather_lsd(
  df,
  l = l,
  s = s,
  d = d,
  bases = c(20, 12),
  lsd_col = lsd,
  replace = FALSE
)

deb_spread_lsd(df, lsd = lsd, l_col = l, s_col = s, d_col = d, replace = FALSE)
```

Arguments

df	A data frame.
l	Pounds column: Unquoted name of a numeric variable corresponding to the pounds or libra unit. Default is l.
s	Shillings column: Unquoted name of a numeric variable corresponding to the shillings or solidus unit. Default is s.
d	Pence column: Unquoted name of a numeric variable corresponding to the pence or denarius unit. Default is d.
bases	Numeric vector of length 2 used to specify the bases for the solidus or s and denarius or d units. Default is c(20, 12), which conforms to the most widely used system of 1 pound = 20 shillings and 1 shilling = 12 pence.
lsd_col	Unquoted name of the deb_lsd column created by the function. Default is lsd.
replace	Logical (default FALSE). When TRUE, the newly created column(s) will replace the one(s) used to create it/them.
lsd	deb_lsd column: Unquoted name of a deb_lsd column. Default is lsd.
l_col	Unquoted name for the pounds column created by the function. Default is l.
s_col	Unquoted name for the shillings column created by the function. Default is s.
d_col	Unquoted name for the pence column created by the function. Default is d.

Details

When transcribing historical accounting data by hand, entering the pounds, shillings, and pence values (lsd) into separate columns is probably the easiest and least error prone method. The deb_gather_() and deb_spread_() set of functions provide helpers to go back and forth between this mode of data entry and the use of deb_lsd and deb_tetra vectors within data frames in R. deb_gather_lsd() creates a deb_lsd column from l, s, and d columns representing the three units of this type of value. deb_spread_lsd() does the opposite. It takes a deb_lsd column and spreads it into three separate pounds, shillings, and pence columns.

Values for column names (lsd_col, l_col, s_col, and d_col) must be valid column names. They can be quoted or unquoted, but they cannot be vectors or bare numbers. This follows the rules of dplyr::rename().

Value

A data frame with a new deb_lsd column for deb_gather_lsd() or new pounds, shillings, and pence columns for deb_spread_lsd().

See Also

deb_gather_tetra() and deb_spread_tetra() provide the same functionality for the less common tetrapartite values of pounds, shillings, pence, and farthings.

Examples

```
libra <- c(3, 5, 6, 2)
solidus <- c(10, 18, 11, 16)
denarius <- c(9, 11, 10, 5)

# data frame with separate l, s, and d variables and default names
x <- data.frame(accounts = c(1, 2, 3, 4),
                l = libra,
                s = solidus,
                d = denarius)

# data frame with deb_lsd variable and default names
y <- data.frame(accounts = c(1, 2, 3, 4),
                lsd = deb_lsd(l = libra,
                              s = solidus,
                              d = denarius))

# Gather l, s, and d variables into deb_lsd column
deb_gather_lsd(x, l = l, s = s, d = d)

# Spread deb_lsd column into separate l, s, and d columns
deb_spread_lsd(y, lsd = lsd)

# Replace original columns with replace = TRUE
deb_gather_lsd(x, replace = TRUE)
deb_spread_lsd(y, replace = TRUE)

# Choose non-default column names
deb_gather_lsd(x, lsd_col = data, replace = TRUE)
deb_spread_lsd(y,
               l_col = libra,
               s_col = solidus,
               d_col = denarius,
               replace = TRUE)

# The two functions are opposites
z <- x %>%
  deb_gather_lsd(replace = TRUE) %>%
  deb_spread_lsd(replace = TRUE)
all.equal(x, z)
```

mathematics *Math group with* deb_lsd *and* deb_tetra *vectors*

Description

Math and Summary group of functions with deb_lsd and deb_tetra vectors. Implemented functions:

- Summary group: sum(), any(), and all().
- Math group: abs(), round(), signif(), ceiling(), floor(), trunc(), cummax(), cummin(), and cumsum().
- Additional generics: mean(), is.nan(), is.finite(), and is.infinite().

All other functions from the groups not currently implemented, including median(), quantile(), and summary().

Arguments

x	An vector of class deb_lsd or deb_tetra.
...	deb_lsd or deb_tetra vectors in sum() and arguments passed on to further methods in mean().
na.rm	Logical. Should missing values (including 'NaN") be removed?
digits	Integer. Indicating the number of decimal places (round()) or significant digits (signif()) to be used.

Details

sum() and cumsum() return a normalized deb_lsd or deb_tetra values.

Round family of functions only affect the denarius (d) unit of a deb_lsd value and the farthing (f) unit of deb_tetra value. All values are normalized.

If you need a wider implementation of Math and Summary group functions, use a deb_decimal vector. However, median(), quantile(), and summary() are also not currently implemented for deb_decimal vectors. To use these functions cast deb_lsd, deb_tetra, and deb_decimal vectors to numeric.

Value

A deb_lsd or deb_tetra vector with normalized values.

Examples

```
x <- deb_lsd(l = c(5, 8, 12),
             s = c(16, 6, 13),
             d = c(6, 11, 0))
y <- deb_tetra(l = c(5, 8, 12),
               s = c(16, 6, 13),
```

```
                d = c(6, 11, 0),
                f = c(3, 2, 3))

# All values are normalized with sum and cumsum
sum(x)
sum(y)
cumsum(x)
cumsum(y)
mean(x)
mean(y)

# Round family on deb_lsd affects the denarius unit
x2 <- deb_lsd(5, 12, 5.8365)
y2 <- deb_tetra(5, 12, 8, 4.125)
round(x2)
round(y2)
round(x2, digits = 2)
signif(x2, digits = 2)
ceiling(x2)
ceiling(y2)
floor(x2)
floor(y2)
trunc(x2)
trunc(y2)

# The returned values are normalized whether
# they are positive or negative
x3 <- deb_lsd(9, 19, 11.825)
x4 <- deb_lsd(-9, -19, -11.825)
round(x3)
round(x3, digits = 1)

ceiling(x3)
floor(x4)

trunc(x3)
trunc(x4)
```

`normalization` *Normalize tripartite and tetrapartite values*

Description

Normalize tripartite and tetrapartite values values to given bases.

Usage

```
deb_normalize(x, ...)
```

```
## Default S3 method:
deb_normalize(x, ...)

## S3 method for class 'deb_lsd'
deb_normalize(x, ...)

## S3 method for class 'numeric'
deb_normalize(x, bases = c(20, 12), ...)

## S3 method for class 'deb_tetra'
deb_normalize(x, ...)
```

Arguments

x	Either an vector of class deb_lsd, deb_tetra, or a numeric vector of length 3 or 4 representing the values to be normalized.
...	Arguments passed on to further methods.
bases	Used only if x is a numeric vector. A Numeric vector of length 2 or 3 used to specify the bases for the solidus or s, denarius or d, and optionally the farthing or f units. Default is c(20, 12), which conforms to the most widely used system of 1 pound = 20 shillings and 1 shilling = 12 pence.

Value

Returns a vector of class deb_lsd with normalized solidus and denarius units or a vector of class deb_tetra with normalized solidus, denarius, and farthing units.

Examples

```
# Normalize a deb_lsd vector
x <- deb_lsd(12, 93, 78)
x_alt <- deb_lsd(12, 93, 78, bases = c(60, 16))
deb_normalize(x)
deb_normalize(x_alt)

# Normalize a deb_tetra vector
t <- deb_tetra(12, 83, 78, 42)
t_alt <- deb_tetra(12, 83, 78, 42, bases = c(60, 16, 8))
deb_normalize(t)
deb_normalize(t_alt)

# Normalize a numeric vector of length 3
deb_normalize(c(12, 93, 78))
deb_normalize(c(12, 93, 78), bases = c(60, 16))

# Normalize a numeric vector of length 4
# Must provide bases of length 3
deb_normalize(c(12, 93, 78, 42), bases = c(20, 12, 4))
deb_normalize(c(12, 93, 78, 42), bases = c(60, 16, 8))
```

tetra-column *Helpers to create and separate a* deb_tetra *column in a data frame*

Description

- deb_gather_tetra() creates a deb_tetra column from separate variables representing pounds, shillings, pence, and farthing values.
- deb_spread_tetra() creates separate variables for pounds, shillings, pence, and farthings from a deb_tetra column.

Usage

```
deb_gather_tetra(
  df,
  l = l,
  s = s,
  d = d,
  f = f,
  bases = c(20, 12, 4),
  tetra_col = tetra,
  replace = FALSE
)

deb_spread_tetra(
  df,
  tetra = tetra,
  l_col = l,
  s_col = s,
  d_col = d,
  f_col = f,
  replace = FALSE
)
```

Arguments

df	A data frame.
l	Pounds column: Unquoted name of a numeric variable corresponding to the pounds or libra unit. Default is l.
s	Shillings column: Unquoted name of numeric variable corresponding to the shillings or solidus unit. Default is s.
d	Pence column: Unquoted name of numeric variable corresponding to the pence or denarius unit. Default is d.
f	Farthing column: Unquoted name of numeric variable corresponding to the farthing or f unit. Default is f.

bases	Numeric vector of length 3 used to specify the bases for the solidus or s, denarius or d, and farthing or f units. Default is `c(20, 12, 4)`, which conforms to the English system of 1 pound = 20 shillings, 1 shilling = 12 pence, and 1 pence = 4 farthing.
tetra_col	Unquoted name of the `deb_tetra` column created by the function. Default is `tetra`.
replace	Logical (default `FALSE`). When `TRUE`, the newly created column(s) will replace the one(s) used to create it/them.
tetra	`deb_tetra` column: Unquoted name of a `deb_tetra` column. Default is `tetra`.
l_col	An unquoted name for the pounds column created by the function. Default is `l`.
s_col	An unquoted name for the shillings column created by the function. Default is `s`.
d_col	An unquoted name for the pence column created by the function. Default is `d`.
f_col	An unquoted name for the farthings column created by the function. Default is `f`.

Details

When transcribing historical accounting data by hand, entering the pounds, shillings, pence, and optionally farthing values (lsd(f)) into separate columns is probably the easiest and least error prone method. The `deb_gather_()` and `deb_spread_()` set of functions provide helpers to go back and forth between this mode of data entry and the use of `deb_lsd` and `deb_tetra` vectors within data frames in R. `deb_gather_tetra()` creates a `deb_tetra` column from four separate columns representing the four units in this type of value. `deb_spread_tetra()` does the opposite. It takes a `deb_tetra` column and spreads it into four separate columns representing the four units.

Values for column names (`tetra_col`, `l_col`, `s_col`, `d_col`, and `f_col`) must be valid column names. They can be quoted or unquoted, but they cannot be vectors or bare numbers. This follows the rules of `dplyr::rename()`.

Value

A data frame with a new `deb_tetra` column for `deb_gather_tetra()` or new pounds, shillings, pence, and farthing columns for `deb_spread_tetra()`.

See Also

`deb_gather_lsd()` and `deb_spread_lsd()` provide the same functionality for the more common tripartite values of pounds, shillings, and pence.

Examples

```
libra <- c(3, 5, 6, 2)
solidus <- c(10, 18, 11, 16)
denarius <- c(9, 11, 10, 5)
farthing <- c(2, 3, 1, 0)

# data frame with separate l, s, and d variables and default names
```

```
x <- data.frame(accounts = c(1, 2, 3, 4),
                l = libra,
                s = solidus,
                d = denarius,
                f = farthing)

# data frame with deb_tetra variable and default names
y <- data.frame(accounts = c(1, 2, 3, 4),
                tetra = deb_tetra(l = libra,
                                  s = solidus,
                                  d = denarius,
                                  f = farthing))

# Gather l, s, d, and f variables into a deb_tetra column
deb_gather_tetra(x, l = l, s = s, d = d, f = f)

# Spread deb_tetra column into separate l, s, d, and f columns
deb_spread_tetra(y, tetra = tetra)

# Replace original columns with replace = TRUE
deb_gather_tetra(x, replace = TRUE)
deb_spread_tetra(y, replace = TRUE)

# Choose non-default column names
deb_gather_tetra(x, tetra_col = data, replace = TRUE)
deb_spread_tetra(y,
                 l_col = libra,
                 s_col = solidus,
                 d_col = denarius,
                 f_col = farthing,
                 replace = TRUE)

# The two functions are opposites
z <- x %>%
  deb_gather_tetra(replace = TRUE) %>%
  deb_spread_tetra(replace = TRUE)
all.equal(x, z)
```

`text` *Format* `deb_lsd`*,* `deb_decimal`*, and* `deb_tetra` *vectors as text*

Description

Flexible way to format `deb_lsd`, `deb_decimal`, and `deb_tetra` vectors for use as labels or text.

Usage

```
deb_text(x, ...)
```

```
## Default S3 method:
deb_text(x, ...)

## S3 method for class 'deb_lsd'
deb_text(
  x,
  digits = 0,
  currency = "£",
  l.mark = "",
  s.mark = "s.",
  d.mark = "d.",
  sep = " ",
  big.mark = ",",
  decimal.mark = ".",
  suffix = "",
  ...
)

## S3 method for class 'deb_decimal'
deb_text(
  x,
  digits = 0,
  currency = "£",
  big.mark = ",",
  decimal.mark = ".",
  suffix = "",
  ...
)

## S3 method for class 'deb_tetra'
deb_text(
  x,
  digits = 0,
  currency = "£",
  l.mark = "",
  s.mark = "s.",
  d.mark = "d.",
  f.mark = "f.",
  sep = " ",
  big.mark = ",",
  decimal.mark = ".",
  suffix = "",
  ...
)
```

Arguments

x	A vector of class deb_lsd, deb_decimal, or deb_tetra.

...	Arguments passed on to further methods.
digits	Desired number of digits after the decimal mark to which to round the numeric values. Default is 0.
currency	Character used for the currency mark. Default is pound sign.
l.mark	Character used following the pounds (l) unit. Default is "".
s.mark	Character used following the shillings (s) unit. Default is "s.".
d.mark	Character used following the pence (d) unit. Default is "d.".
sep	Character to separate pounds, shillings, pence, and optionally farthing units. Default is " ".
big.mark	Character used to mark intervals to the left of the decimal mark. Default is "," with default big.interval of 3.
decimal.mark	Character used for decimal mark. Default is ".".
suffix	Character placed after the values. Default is "".
f.mark	Character used following the farthing (f) unit with tetrapartite values. Default is "f.".

Details

deb_text is similar to as.character() in that both return a character vector of the values of deb_lsd, deb_decimal, and deb_tetra vectors. However, as.character() uses the normal printing method for these vectors. deb_text() provides a convenient way to nicely format deb_lsd, deb_decimal, and deb_tetra vectors for use as text or labels with options for customization.

deb_text() uses formatC() to format the numeric values of x. Numbers are printed in non-scientific format and trailing zeros are dropped.

All character vector arguments should be length 1.

Value

A Character vector of formatted values.

See Also

formatC() for further options passed to ....

Examples

```
lsd <- deb_lsd(l = c(10000, 0, -10000),
               s = c(8, 0, -8),
               d = c(5.8252, 0, -5.8252))
dec <- deb_decimal(c(10000.8252, 0, -10000.8252))
tetra <- deb_tetra(l = c(10000, 0, -10000),
                   s = c(8, 0, -8),
                   d = c(5, 0, -5),
                   f = c(2.8252, 0, -2.8252))

deb_text(lsd)
deb_text(dec)
```

```
deb_text(tetra)

# Compact format for deb_lsd with suffix to distinguish currency
deb_text(lsd, s.mark = "", d.mark = "",
         sep = ".", suffix = " Flemish")

# Control the number of digits
deb_text(lsd, digits = 3)
deb_text(dec, digits = 3)
deb_text(tetra, digits = 3)

# Change big mark and decimal mark
deb_text(lsd, digits = 4, big.mark = ".", decimal.mark = ",")
deb_text(dec, digits = 4, big.mark = ".", decimal.mark = ",")
deb_text(tetra, digits = 4, big.mark = ".", decimal.mark = ",")
```

`transactions` *Analysis of double-entry bookkeeping*

Description

Family of seven related functions to analyze transactions data frames that have credit, debit, and tetrapartite (lsd) or tetrapartite (lsdf) columns, mimicking an account book.

- `deb_account()` credit, debit, and current value of a single account.
- `deb_account_summary()` credit, debit, and current value of all accounts.
- `deb_credit()` total credit of each account.
- `deb_debit()` total debit of each account.
- `deb_current()` current value of each account (credit - debit).
- `deb_open()` current value of each account that has a positive or negative value.
- `deb_balance()` positive and negative value remaining in a transactions data frame.

Usage

```
deb_account(
  df,
  account_id,
  credit = credit,
  debit = debit,
  lsd = lsd,
  na.rm = FALSE
)

deb_account_summary(
  df,
  credit = credit,
```

```
  debit = debit,
  lsd = lsd,
  na.rm = FALSE
)

deb_credit(df, credit = credit, debit = debit, lsd = lsd, na.rm = FALSE)

deb_debit(df, credit = credit, debit = debit, lsd = lsd, na.rm = FALSE)

deb_current(df, credit = credit, debit = debit, lsd = lsd, na.rm = FALSE)

deb_open(df, credit = credit, debit = debit, lsd = lsd, na.rm = FALSE)

deb_balance(df, credit = credit, debit = debit, lsd = lsd, na.rm = FALSE)
```

Arguments

df	A data frame or tibble with at least credit, debit, and lsd columns.
account_id	The id of the account to be used to calculate the credit, debit, and current values.
credit	Credit column: Unquoted name of the credit column, representing the accounts that discharge the transactional values or from which the values derive. Default is credit.
debit	Debit column: Unquoted name of the debit column, representing the accounts that receive the transactional values. Default is debit.
lsd	Value column: Unquoted name of a column of class deb_lsd, deb_decimal, or deb_tetra. Default is lsd.
na.rm	Logical. Should missing values (including NaN) be removed?

Value

Transaction functions return a data frame or tibble with columns for the accounts in df and credit, debit, and/or current values in the same type and bases as lsd:

- deb_account(): a data frame with three rows showing the credit, debit, and current value of the given account.
- deb_account_summary() a data frame with one row for each account in df and credit, debit, and current value columns.
- deb_credit(): a data frame with one row for each account with the total credit of the accounts.
- deb_debit(): a data frame with one row for each account with the total debit of the accounts.
- deb_current(): a data frame with one row for each account with the current value of the accounts.
- deb_open(): a data frame with one row for each account whose current value is not 0. If all accounts are equal to zero, a data frame with zero rows will be returned.
- deb_balance(): a data frame with two rows showing the credit and debit remaining in df.

Transactions data frames:

Transactions data frames have the structure of an account book. They should have a similar arrangement to `dafforne_transactions`. Each row is a transaction in the book. `credit` and `debit` columns contain the account ids associated with discharging account (credit) and the receiving account (debit). The `lsd` column represents the tripartite or tetrapartite value of each transaction. Like `dafforne_transactions`, transactions data frames can have additional columns with attributes for each transaction such as id or date among others.

Examples

```
# Examples use dafforne_transactions data,
# which have default column names.
# See dafforne_accounts for account names.

# Credit, debit, and current value of cash account
deb_account(dafforne_transactions, account_id = 1,
            credit = credit, debit = debit,
            lsd = lsd)

# Credit, debit, and current value of profit and loss account
deb_account(dafforne_transactions, account_id = 23)

# Summary of all accounts in Dafforne's ledger
deb_account_summary(dafforne_transactions)

# Credit of accounts in Dafforne's ledger
deb_credit(dafforne_transactions)

# Debit of accounts in Dafforne's ledger
deb_debit(dafforne_transactions)

# Current value of accounts in Dafforne's ledger
current <- deb_current(dafforne_transactions)
current

# Current value of open account in Dafforne's ledger
open <- deb_open(dafforne_transactions)
open

# Compare the amount of rows in returned values of
# deb_current() vs deb_open()
nrow(current)
nrow(open)

# Credit and debit remaining on Dafforne's ledger
deb_balance(dafforne_transactions)
```

vec_math.deb_lsd *Error message for unimplemented mathematics functions*

Description

Error message for unimplemented mathematics functions

Usage

```
## S3 method for class 'deb_lsd'
vec_math(.fn, .x, ...)
```

Arguments

.fn	A mathematical function from the base package.
.x	A vector.
...	Additional arguments passed to .fn.

Value

A deb_lsd vector.

vec_math.deb_tetra *Error message for unimplemented mathematics functions*

Description

Error message for unimplemented mathematics functions

Usage

```
## S3 method for class 'deb_tetra'
vec_math(.fn, .x, ...)
```

Arguments

.fn	A mathematical function from the base package.
.x	A vector.
...	Additional arguments passed to .fn.

Value

A deb_tetra vector.

Index