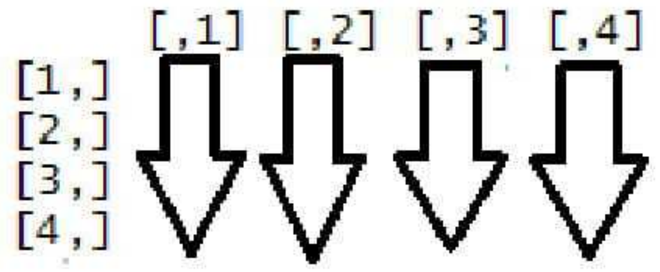
Apply Family functions

Apply

The `apply()` function is an alternative to writing loops, via applying a function to columns, rows, or individual values of an array or matrix.



The structure of the `apply()` function is:
`apply(X, MARGIN, FUN, ...)`

The matrix variable used for the exercises is:
`dataset1 <- cbind(observationA = 16:8, observationB = c(20:19, 6:12))`

Exercise 1

Using `apply()`, find the row means of `dataset1`

Exercise 2

Using `apply()`, find the column sums of `dataset1`

Exercise 3

Use `apply()` to sort the columns of `dataset1`

Exercise 4

Using `apply()`, find the product of `dataset1` rows

Exercise 5

Required function:
`DerivativeFunction <- function(x) { log10(x) + 1 }`

Apply "`DerivativeFunction`" on the rows of `dataset1`

Exercise 6

Re-script the formula from Exercise 5, in order to define '`DerivativeFunction`" inside the `apply()` function

Exercise 7

Round the output of the Exercise 6 formula to 2 places

Exercise 8

Print the columns of `dataset1` with the `apply()` function

Exercise 9

Find the length of the `dataset1` columns

The `lapply()` function applies a function to individual values of a list, and is a faster alternative to writing loops.

Structure of the `lapply()` function:
`lapply(LIST, FUNCTION, ...)`

The list variable used for these exercises:
`list1 <- list(observationA = c(1:5, 7:3), observationB=matrix(1:6, nrow=2))`

Exercise 1

Using `lapply()`, find the length of `list1` 's observations.

Exercise 2

Using `lapply()`, find the sums of `list1` 's observations.

Exercise 3

Use `lapply()` to find the quantiles of `list1`.

Exercise 4

Find the classes of `list1` 's sub-variables, with `lapply()`.

Exercise 5

Required function:
`DerivativeFunction <- function(x) { log10(x) + 1 }`

Apply the "`DerivativeFunction`" to `list1`.

Exercise 6

Script the "`DerivativeFunction`" within `lapply()`. The dataset is `list1`.

Exercise 7

Find the unique values in `list1`.

Exercise 8

Find the range of `list1`.

Exercise 9

Print `list1` with the `lapply()` function.

Exercise 10

`mapply()` works with multivariate arrays, and applys a function to a set of vector or list arguments. `mapply()` also simplifies the output.

Structure of the `mapply()` function:

```
mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

Exercise 1

Beginning level

Required dataframe:

```
PersonnelData <- data.frame(Representative=c(1:4),
Sales=c(95,110,115,90), Territory=c(1:4))
```

Using `mapply()`, find the classes of `PersonnelData` 's columns.

Exercise 2

Beginning level

Print " `PersonnelData` " with the `mapply()` function.

Exercise 3

Beginning level

Use `mapply()` to inspect " `PersonnelData` " for numeric values.

Exercise 4

Intermediate level

Use `mapply()` to sum the vectors " `5:10` " and " `20:25` ".

Exercise 5

Intermediate level

Use `mapply()` to paste the vector " `1:4` " and " `5:8` ", with the separator " `LETTERS[1:4]` ".

Exercise 6

Intermediate level

Use `mapply()` to paste " `PersonnelData$Representative` ", " `PersonnelData$Sales` ", and " `PersonnelData$Territory` ", with the " `MoreArgs=` " argument of " `list(sep="-")` ".

Exercise 7

Advanced level

Required variable:
`NewSales <- data.frame(Representative=c(1:4), Sales=c(104, 97, 112, 94), Territory=c(1:4))`

Sum the corresponding elements of `PersonnelData$Sales` and `NewSales$Sales`.

Exercise 8
Advanced level

Required function:
`merge.function <- function(x,y){return(x+y)}`

Use `merge.function` to combine the `Sales` totals from `PersonnelData` and `NewSales`.

Exercise 9
Advanced level

`mcmapply` is a parallelized version of `mapply`.

The structure of `mcmapply()` is:
`mcmapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE, mc.preschedule = TRUE, mc.set.seed = TRUE, mc.silent = FALSE, mc.cores = getOption("mc.cores", 2L), mc.cleanup = TRUE)`

Required library:
`library(parallel)`

Use `mcmapply()` to generate 5 lists of `1:5` random numbers.

Using `mcmapply()`, create a 10 by 10 matrix with 10 rows of the sequence `1:10`: