The very basics

Data types

R has three basic data types:

1(a). **Numeric** ("real numbers").

⇒ The two most common numeric types are `double` (double precision floating point numbers) and `integer` (without floating point).

1(b). **Complex** ("real and imaginary numbers").

2. **Logical** ("boolean").

⇒ Reserved words for denoting logical constants are `TRUE` and `FALSE` (and `NA` for missing value).

3. **Character**.

⇒ Data type for storing letters and symbols (strings, text).

Data structures

1. **Scalar**.
2. **Vector**. Collection of elements of a single ("atomic") data type.
3. **Matrix**. Collection of elements arranged in a two-dimensional rectangular layout (a two-dimensional generalization of a vector). Same as vector, all elements must be of a single data type.
4. **Data frame**. More general matrix like structure ("data matrix"). Different columns can have different data types.
5. **List**. Generic vector containing other objects. No restriction on data types or length of the single components.

Vectors

Concatinating elements together with `c()`.

```
> c(0.5, 0.6, 0.25)            # double
> c(9L, 10L, 11L, 12L, 13L)    # integer
> c(9:13)                      # integer sequence
> c(TRUE, FALSE, FALSE)        # logical
> c(1+0i, 2+4i)                # complex
> c("a", "b", "c")             # character
```

Vector actions

Assign the vectors to names:

```
> dbl <- c(0.5, 0.6, 0.25)
> chr <- c("a", "b", "c")
```

Print out the `dbl` and `chr` vectors on the console:

```
> dbl
```

```
[1] 0.50 0.60 0.25
```

```
> chr
```

```
[1] "a" "b" "c"
```

Check the number of elements in `dbl` and `chr`:

```
> length(dbl)
```

```
[1] 3
```

```
> length(chr)
```

```
[1] 3
```

Check the data type `dbl` and `chr`:

```
> typeof(dbl)
```

```
[1] "double"
```

```
> typeof(chr)
```

```
[1] "character"
```

Combine two vectors:

```
> c(dbl,dbl)
```

```
[1] 0.50 0.60 0.25 0.50 0.60 0.25
```

```
> c(dbl, chr)
```

```
[1] "0.5"  "0.6"  "0.25" "a"    "b"    "c"
```

! The automatic change of the data type of the resulting vector is called **coercion**. Coercion ensures the same data type for each element in the vector is maintained.

Vector arithmetic

Define two new numeric vectors `a` and `b` each having 4 elements:

```
> a <- c(1, 2, 3, 4)
> b <- c(10, 20, 30, 40)
```

Multiply each element in `a` by 5 (scalar multiplication):

```
> a * 5
```

```
[1]  5 10 15 20
```

Multiply the elements in `a` by the elements in `b` (vector multiplication):

```
> a * b
```

```
[1]  10  40  90 160
```

Multiply the elements in `a` by the elements of some numeric vector `v` of length 5:

```
> v <- c(1.1, 1.2, 1.3, 1.4, 1.5)
> a * v
```

```
Warning in a * v: Länge des längeren Objektes
     ist kein Vielfaches der Länge des kürzeren Objektes
```

```
[1] 1.1 2.4 3.9 5.6 1.5
```

! Arithmetic operations of vectors are performed **elementwise**. If two vectors are of unequal length, the shorter vector will be **recycled** in order to match the longer one (here, the first element in `a` is used again).

Matrices

Option (1): Combining two vectors columnwise with `cbind()`:

```
> A <- cbind(a, b)    # two columns
> A
```

```
     a  b
[1,] 1 10
[2,] 2 20
[3,] 3 30
[4,] 4 40
```

Option (2): Combining two vectors rowwise with `rbind()`:

```
> B <- rbind(a, b)    # two rows
> B
```

```
  [,1] [,2] [,3] [,4]
a    1    2    3    4
b   10   20   30   40
```

Option (3): Creating a matrix from elements of a vector with `matrix()`:

```
> A <- matrix(a, ncol=2, nrow=2)     # matrix with 2 columns and 2 rows
> A
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

The arguments *nrow* and *ncol* indicate the number of rows and number of columns the resulting matrix consists of.

! For 4 elements and *ncol* = 2 the matrix can only have 2 rows. Thus, there is no need to specify both arguments.

```
> A <- matrix(a, ncol=2)              # matrix with 2 columns and 2 rows
> A
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

! By default the matrix is filled up column after column (R treats a matrix object internally as a column vector). If the matrix should be filled up row after row the argument *byrow=* TRUE is required.

```
> B <- matrix(a, ncol=2, byrow=TRUE) # matrix filled-up rowwise
> B
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Matrix actions

Checking the number of rows:

```
> nrow(B)
```

```
[1] 2
```

Checking the number of columns:

```
> ncol(B)
```

```
[1] 2
```

Checking the dimension [nrow, ncol]:

```
> dim(B)
```

```
[1] 2 2
```

Combine two matrices:

```
> D.wide <- cbind(A,A)
> D.wide
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    3
[2,]    2    4    2    4
```

```
> D.long <- rbind(A,A)
> D.long
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    3
[4,]    2    4
```

```
D <- cbind(D.wide, D.long)
```

```
Error in cbind(D.wide, D.long) : object 'D.wide' not found
```

❗ Two matrices of unequal dimensions (number of rows or number of columns) cannot be combined.

Matrix arithmetic

Matrix addition:

```
> B + B
```

```
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

Scalar multiplication:

```
> B * 2
```

```
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

Elementwise multiplication:

```
> B * B
```

```
     [,1] [,2]
[1,]    1    4
[2,]    9   16
```

Matrix multiplication:

```
> B %*% B
```

```
     [,1] [,2]
[1,]    7   10
[2,]   15   22
```

More matrix arithmetic

- Transpose **t()**

```
> D.wide
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    3
[2,]    2    4    2    4
```

```
> t(D.wide)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    1    2
[4,]    3    4
```

- Determinant **det()**

```
> det(B)
```

```
[1] -2
```

- Inverse **solve()** (only if **det()** $\neq 0$)

```
> solve(B)
```

```
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

- Eigenvalues **eigen()** (only for square and symmetric matrices)

```
> eigen(B)
```

```
eigen() decomposition
$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.4159736 -0.8245648
[2,] -0.9093767  0.5657675
```

Data frames

```
> dbl <- c(0.5, 0.6, 0.25, 1.2, 0.333)      # double
> int <- c(9L, 10L, 11L, 12L, 13L)          # integer
> lgl <- c(TRUE, FALSE, FALSE, TRUE, TRUE)  # logical
> chr <- c("a", "b", "c", "d", "e")         # character
> df <- data.frame(dbl,int,lgl,chr)
> df
```

```
    dbl int   lgl chr
1 0.500   9  TRUE   a
2 0.600  10 FALSE   b
3 0.250  11 FALSE   c
4 1.200  12  TRUE   d
5 0.333  13  TRUE   e
```

Data frame actions

Checking the number of rows:

```
> nrow(df)
```

```
[1] 5
```

Checking the number of columns:

```
> ncol(df)
```

```
[1] 4
```

Checking the dimension `[nrow, ncol]`:

```
> dim(df)
```

```
[1] 5 4
```

Lists

```
> a <- 1L                                   # scalar
> dbl <- c(0.5, 0.6, 0.25, 1.2, 0.333)      # numeric vector of length 5
> chr <- c("a", "b", "c"        )           # charcter vector of length 3
> v <- c(1.1, 1.2, 1.3, 1.4)
> mat <- matrix(v, ncol=2)                  # 2 x 2 matrix
>
```

```
> l <- list(a, dbl, chr, mat)
> l
```

```
[[1]]
[1] 1

[[2]]
[1] 0.500 0.600 0.250 1.200 0.333

[[3]]
[1] "a" "b" "c"

[[4]]
     [,1] [,2]
[1,]  1.1  1.3
[2,]  1.2  1.4
```