Lists

- Another basic structure in R is a list. The main advantage of lists is that the “columns” (they’re not really ordered in columns any more, but are more a collection of vectors) don’t have to be of the same length, unlike matrices or data frames (see below). Actually a list is a matrix where two rules are over-written: i) In every column you can have a different type of data, e.g. in column one integers, and in column two characters. ii) not all the columns need to have same length. The following lines construct a list by giving names and values. This list should appear in your work space as well.

```r
> L <- list(first="A", second=c("male", "female"), third=seq(0, 1, length=5))
> L
$first
 [1] "A"
$second
 [1] "male" "female"
$third
 [1] 0.00 0.25 0.50 0.75 1.00

> names(L)
[1] "first" "second" "third"
```

- Now let’s see an example of how to work with lists. If we want to increase the values of the numbers in this list:

```r
> L$third + 1
[1] 1.00 1.25 1.50 1.75 2.00
```

Notice how we access the elements in a list using the "$" symbol.

Data frames

- All elements of a matrix must be of the same mode (numeric, character, logical, etc.). If you try to put different modes in a matrix, all elements will be coerced to the most general—usually the character mode.

```r
> Person <- c("Bob", "Bill", "Betty")
> TestA <- c(80, 95, 92)
> TestB <- c(40, 87, 90)
> grades <- cbind(Person, TestA, TestB)
> grades
   Person TestA TestB
[1,] "Bob" "80" "40"
[2,] "Bill" "95" "87"
[3,] "Betty" "92" "90"
```

Note that all numbers are now interpreted as character strings. That is usually not what you want.
• The solution to this problem is another complex object called a *data frame*. The data frame views rows as cases and columns as variables. All elements in a column must be of the same mode, but different columns may be of different modes.

```r
# Create a data frame from vectors:
> grades.df <- data.frame(Person, TestA, TestB)
> grades.df

Person TestA TestB
1  Bob  80  40
2  Bill 95  87
3 Betty 92  90
```

• Note that the columns of data frames have *names*. These can be used to extract values from the data frame.

```r
> grades.df$TestB
[1] 40 87 90
```

```r
> grades.df$Person[2]
[1] Bill
Levels: Betty Bill Bob
```

You can change the names of the columns:

```r
> colnames(grades.df) <- c("Name", "Test1", "Test2")
> grades.df

Name Test1 Test2
1 Bob  80  40
2 Bill 95  87
3 Betty 92  90
```

• The function `str()` returns the structure of any R object.

```r
# Name is a factor with 3 levels, Test1 and Test2 are numerical vectors.
> str(grades.df)

'data.frame': 3 obs. of 3 variables:
$ Name : Factor w/ 3 levels "Betty","Bill",..: 3 2 1
$ Test1: num 80 95 92
$ Test2: num 40 87 90
```

The matrix created with `cbind()` has a different structure: all data are converted to character type.

```r
> str(grades)  # structure of the matrix created with cbind.

chr [1:3, 1:3] "Bob" "Bill" "Betty" "80" ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:3] "Person" "TestA" "TestB"
```

• Within RStudio, you can also view data frames as follows:

```r
> View(grades.df)
```

Try it!
Summary functions applied to a data frame will be applied to each column. This may not always work:

```r
> mean(grades.df)
[1] NA
Warning message:
In mean.default(grades.df) :
  argument is not numeric or logical: returning NA
```

Note that there is a problem: the mean of the column `Name` is obviously undefined. R returns the value `NA` for “not available”. You may want to specify the column of interest:

```r
> mean(grades.df$Test1)
[1] 89
```

**NAs** have to be carefully handled. For example, let’s include one NA in the previous data frame.

```r
> grades.df[2,3] <- NA
> grades.df
   Name  Test1  Test2
1  Bob   80   40
2  Bill  95    NA
3 Betty  92   90
```

Now, R cannot compute the mean of the `Test2` column:

```r
> mean(grades.df$Test2)
[1] NA
```

Therefore, the result is another NA. You can tell R to remove the NA values:

```r
> mean(grades.df$Test2, na.rm = TRUE)
[1] 65
```

Note that the result is the mean of the remaining values.

Continuous variables can be converted into factors as follows:

```r
> grades.df$Test1 <- as.factor(grades.df$Test1)
> grades.df$Test1
 [1] 80 95 92
Levels: 80 92 95
```

# Calling the structure reveals this change: Test1 is now a factor with 3 levels.
```

```r
> str(grades.df)
'data.frame': 3 obs. of 3 variables:
$ Name : Factor w/ 3 levels "Betty","Bill",...: 3 2 1
$ Test1: Factor w/ 3 levels "80","92","95": 1 3 2
$ Test2: num 40 NA 90
```