# Lists, Matrices and Arrays

# Data Structures

- vectors
- factors
- *lists*
- matrices
- arrays
- dataframes

# Lists

Lists are flexible structures that can contain objects of different types and different sizes (each element can itself be a list), and they are created with the `list()` command

```
my_list <- list("hello!",
                1:3,
                c(TRUE,FALSE),
               list("hello!",
                    1:3,
                    c(TRUE,FALSE)))
```

# Lists

List develop in depth…

```
1 str(my_list)
```

```
List of 4
 $ : chr "hello!"
 $ : int [1:3] 1 2 3
 $ : logi [1:2] TRUE FALSE
 $ :List of 3
  ..$ : chr "hello!"
  ..$ : int [1:3] 1 2 3
  ..$ : logi [1:2] TRUE FALSE
```

# Lists

We can assign names to the elements of a list…

```
my_list_nam <- list(word = "hello!",
                    numbers =  1:3,
                    logical = c(TRUE,FALSE),
                    list = list("hello!", 1:3, c(TRUE,FALSE)))
```

# Lists

We can assign names to the elements of a list…

```
1 my_list_nam
```

```
$word
[1] "hello!"

$numbers
[1] 1 2 3

$logical
[1]  TRUE FALSE

$list
$list[[1]]
[1] "hello!"
```

# Lists

and access them by calling them by “name” with the `$` operator

```
my_list_nam$word
```

```
[1] "hello!"
```

```
my_list_nam$list # list created as an element of the list
```

```
[[1]]
[1] "hello!"

[[2]]
[1] 1 2 3

[[3]]
[1]  TRUE FALSE
```

# Lists - Attributes

Like vectors, lists also have a length (**`length()`**) and possibly names (**`names()`**).

```
1 length(my_list_nam)
```

```
[1] 4
```

```
1 names(my_list_nam)
```

```
[1] "word"    "numbers" "logical" "list"
```

The **`str()`** (structure) command is very useful for lists because it gives an overview of their structure:

```
1 str(my_list_nam)
```

```
List of 4
 $ word   : chr "hello!"
 $ numbers: int [1:3] 1 2 3
 $ logical: logi [1:2] TRUE FALSE
 $ list   :List of 3
  ..$ : chr "hello!"
  ..$ : int [1:3] 1 2 3
  ..$ : logi [1:2] TRUE FALSE
```

# Lists - Indexing

The difference between the square brackets is whether we want to subset the list, getting another list, or directly access the inner element:

```
1 my_list_nam[1] # getting another list
```

```
$word
[1] "hello!"
```

```
1 my_list_nam[[1]] # accessing the inner element
```

```
[1] "hello!"
```

# Lists - Indexing

If we want to actually subset the list, we must always use single square brackets:

```r
# list made of the first two elements of my_list_nam
my_list2 <- my_list_nam[1:2]
str(my_list2)
```

```
List of 2
 $ word   : chr "hello!"
 $ numbers: int [1:3] 1 2 3
```

# Lists - Nested Indexing

Lists have a one-dimensional structure that can develop in depth. To select nested elements we can chain several brackets:

```
1 my_list_nam[2]
```

```
$numbers
[1] 1 2 3
```

```
1 my_list_nam[[2]][1]
```

```
[1] 1
```

```
1 my_list_nam[2][1] # wrong!
```

```
$numbers
[1] 1 2 3
```

# Lists - Why?

They are flexible and allow us to store/organize complex and varied data/information. For example, the `lm()` function returns a list containing various pieces of information and results about the tested model.

```r
# Simulate 30 observations, assuming that the effect of x = 0.3
N <- 30; b1 <- 0.3; sigma <- 1
x <- rnorm(N, 0, 1); # independent variable
y <- rnorm(n = N, mean = b1*x, sd = sigma)
```

```r
# linear model
mod <- lm(y ~ x)
summary(mod)
```

```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.74636 -0.72466 -0.04939  0.77853  2.03866 
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09571    0.19741   0.485    0.632
x            0.01237    0.17956   0.069    0.946
```

# Lists - Why?

```r
# what is the mod object?
str(mod)
```

```
List of 12
 $ coefficients : Named num [1:2] 0.0957 0.0124
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
 $ residuals    : Named num [1:30] 0.913 1.287 1.69 -0.475 -1.746 ...
  ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
 $ effects      : Named num [1:30] -0.549 0.0702 1.889 -0.8189 -1.7025
...
  ..- attr(*, "names")= chr [1:30] "(Intercept)" "x" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:30] 0.1114 0.0859 0.1275 0.0867 0.1159
...
  ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
```

# Data Structures

- vectors
- factors
- lists
- ***matrices***
- arrays
- dataframes

# Matrices

Matrices are a **two-dimensional** data structure (characterized by 2 dimensions, `dim()`), where the number of rows is dimension 1 and the number of columns is dimension 2.

```
1 my_mat <- matrix(data = 1:10, nrow = 2, ncol = 5, byrow = FALSE)
2 my_mat
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

```
1 matrix(data = 1:10, nrow = 2, ncol = 5, byrow = TRUE)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

# Matrices - Attributes

```r
attributes(my_mat)
```

```
$dim
[1] 2 5
```

```r
my_mat <- matrix(data = 1:10, nrow = 2,
                 ncol = 5, dimnames = list(c("row1", "row2"),
                                           c("col1", "col1",
                                             "col3","col4","col5")))
attributes(my_mat)
```

```
$dim
[1] 2 5

$dimnames
$dimnames[[1]]
[1] "row1" "row2"

$dimnames[[2]]
[1] "col1" "col1" "col3" "col4" "col5"
```

# Matrices - Properties

- They can contain **only one type** of data
- Since they are **two-dimensional**, we need two position indices (rows and columns) to identify an element
- They can be seen as a **set** of single **vectors**

# Matrices - Properties

The number of rows and columns does not have to be the same (square matrix), but the size of the matrix must be compatible with the data vector:

```r
matrix(data = 1:10, ncol = 3, nrow = 3)
```

```
Warning in matrix(data = 1:10, ncol = 3, nrow = 3): data length [10] is
not a
sub-multiple or multiple of the number of rows [3]
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

# Matrices

What does R do by default?

```r
matrix(data = 1:10, ncol = 3, nrow = 3)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```r
matrix(data = 1:2, ncol = 3, nrow = 3)
```

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

**warnings**: the function informs us about something potentially problematic, but (more or less!!) everything is fine

# Matrices - Indexing

To identify one or more elements of a matrix we need row and/or column indices separated by a comma, again inside square brackets: **matrix[row, column]**

```
1 my_mat
```

```
     col1 col1 col3 col4 col5
row1    1    3    5    7    9
row2    2    4    6    8   10
```

```
1 my_mat[1,1]
```

```
[1] 1
```

# Matrices - Indexing

We can also select an entire row or column

```
1 my_mat[1,]
```

```
col1 col1 col3 col4 col5
   1    3    5    7    9
```

```
1 my_mat[,1]
```

```
row1 row2
   1    2
```

# Matrices - Indexing

As with vectors, here too we can apply logical indexing:

```
1 my_mat>2
```

```
      col1 col1 col3 col4 col5
row1 FALSE TRUE TRUE TRUE TRUE
row2 FALSE TRUE TRUE TRUE TRUE
```

```
1 my_mat
```

```
     col1 col1 col3 col4 col5
row1    1    3    5    7    9
row2    2    4    6    8   10
```

```
1 # All the elements greater than two
2 my_mat[my_mat>2]
```

```
[1]  3  4  5  6  7  8  9 10
```

# Vectors and Matrices

Vectors are created with the `c()` function and can be concatenated using the same function:

```
my_vect1 <- c(1:4)
my_vect2 <- c(5:10)

my_vect12 <- c(my_vect1,my_vect2)
my_vect12
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

# Matrices

Matrices can be joined together with the commands **cbind()** or **rbind()**

```
1 my_mat1 <- matrix(data = 1:4,nrow = 2, ncol = 2)
2 my_mat1
```

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

```
1 my_mat2 <- matrix(data = letters[1:4],nrow = 2, ncol = 2)
2 my_mat2
```

```
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

# Matrices

## cbind()

```
1 cbind(my_mat1, my_mat2)
```

```
     [,1] [,2] [,3] [,4]
[1,] "1"  "3"  "a"  "c"
[2,] "2"  "4"  "b"  "d"
```

## rbind()

```
1 rbind(my_mat1, my_mat2)
```

```
     [,1] [,2]
[1,] "1"  "3"
[2,] "2"  "4"
[3,] "a"  "c"
[4,] "b"  "d"
```

Do you notice anything “strange”?

# Operations with matrices

As with vectors, mathematical operations can also be applied to matrices:

```
my_mat <- matrix(data = 1:4,nrow = 2, ncol = 2)

# element-wise
my_mat*my_mat
```

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

```
# Matrix product # (1*1 + 3*2) , (1*3 + 3*4)
my_mat%*%my_mat  # (2*1 + 4*2) , (2*3 + 4*4)
```

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

# Data Structures

- vectors
- factors
- lists
- matrices
- ***arrays***
- dataframes

# Arrays

Arrays are n-dimensional objects (e.g., if a matrix is a square, an array is a cube). The same properties of matrices apply, scaled to n dimensions; they are created with the `array` command

```
1 my_array <- array(data = 1:6, dim = c(2,3,2)) # three-dimensional ex
2 my_array
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

# Arrays - Indexing

Indexing works the same way as with matrices, adding one dimension

```
my_array[1,1,1] # first row, first column, first "slice"
```

```
[1] 1
```

```
my_array[1,2,1] # first row, second column, first "slice"
```

```
[1] 3
```

# Let’s practice!

Open this link and keep it open:

https://etherpad.wikimedia.org/p/arca-corsoR