# RAdvanced
## Lists

PhD Toolbox - R introductory course - stream II
Martino ADAMO
Matteo CHIALVA

## object class:list

A list is a group of any type of objects (vector, matrix, data.frame)

- function is **list**(), each object in the list can be identified by a name

```
> vector <- rep(c("A", "B"), 2, each=2)

> matrix <- rbind(c(1,2),c(1,2),c(1,2))

> df<-data.frame(  "colname_1"=rep(c("A", "B"), 2, each=2),
                   "colname_2"=sample(seq(1,30), 8),
                   "colname_3"=seq(8,11.5, by=0.5))


> my_list<-list("vector"=vector, "matrix"=matrix, "dataframe"=df)

> str(my_list)
List of 3
 $ vector   : chr [1:8] "A" "A" "B" "B" ...
 $ matrix   : num [1:3, 1:2] 1 1 1 2 2 2
 $ dataframe:'data.frame':8 obs. of  3 variables:
  ..$ colname_1: chr [1:8] "A" "A" "B" "B" ...
  ..$ colname_2: int [1:8] 7 24 20 23 22 25 1 10
  ..$ colname_3: num [1:8] 8 8.5 9 9.5 10 10.5 11 11.5
```

## rows/columns/object indexing in lists

select element in list

```
> my_list[[1]]  # positional selection (select the first element)

[1] "A" "A" "B" "B" "A" "A" "B" "B"


> my_list[["df"]] # selection by name
  colname_1 colname_2 colname_3
1         A        13       8.0
2         A         5       8.5
3         B        20       9.0
4         B        21       9.5
5         A         9      10.0
6         A        28      10.5
7         B         8      11.0
8         B        12      11.5
```

# rows/columns/object indexing (lists)

select column in a dataframe included in a list

```
> my_list[["df"]]$colname_1
[1] "A" "A" "B" "B" "A" "A" "B" "B"
```

select rows (or columns) in element included in a list

```
> my_list[["df"]][1:5,]
  colname_1 colname_2 colname_3
1         A        13       8.0
2         A         5       8.5
3         B        20       9.0
4         B        21       9.5
5         A         9      10.0
```

## Why to work with lists in R

Few....but worth reasons!

- reduce lines of code
- Enhance reproducibility in your data analysis

    functions are applied simultaneously to all list elements

- speed-up complex data analysis (multi-core functions)

## functions to work with lists

Main functions are:

- **list()**
- lapply()
- sapply
- unlist()
- do.call() / rbindlist

# lapply()          list-apply

We first need to introduce the **apply**() function

**What it does?**

Takes a matrix or data.frame in input and applies functions to it **column-wise** or **row-wise**

```
> m1 <- matrix(1:10,nrow=5, ncol=6)

> m1

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

## apply()

We first need to introduce the **apply**() function

**apply**( object, margin, function )

```
# margin=1 applies function by rows

> apply(m1, 1, sum)
[1] 21 27 33 39 45

# margin=2 applies function by rows

> apply(m1, 2, sum)
[1] 15 40 15 40 15 40
```

## apply()

We first need to introduce the **apply**() function

**apply**( object, margin, function )

```
# with custom functions
> apply(m1, 2, function(x) x+1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    2    7    2    7    2    7
[2,]    3    8    3    8    3    8
[3,]    4    9    4    9    4    9
[4,]    5   10    5   10    5   10
[5,]    6   11    6   11    6   11
```

**Note: apply() returns a vector or a matrix/dataframe depending on the input and the function applied.**

# lapply()            list-apply

## What it does?

Applies a function to list/vector of elements. Functions are applied to each element individually.

**lapply**(object, function)

```
# a simple example on a vector

> vector<-c(1,2,3,4)

> lapply_output<-lapply(vector, function(x) x+100)

> str(lapply_output)
List of 4
 $ : num 101
 $ : num 102
 $ : num 103
 $ : num 104


> unlist(lapply output)
[1] 101 102 103 104
```

## lapply()            list-apply

**What it does?**

Applies a function to list/vector of elements. Functions are applied to each element individually.

**lapply**(object, function)

```
# a simple example on list

# generate a list from two simple vectors
> fruits=c("banana", "apple", "apricot", "peach")
> colors=c("yellow", "red", "orange", "pink")

> list<-list(fruits=fruits,colors=colors)

> list
$fruits
[1] "banana"  "apple"   "apricot" "peach"

$colors
[1] "yellow" "red"    "orange" "pink"
```

## lapply()            list-apply

**What it does?**

Applies a function to list/vector of elements. Functions are applied to each element individually.

**lapply**(object, function)

```
# a simple example on list

> lapply(list, length)
$fruits
[1] 4


$colors
[1] 4
```

## lapply()          list-apply

**What it does?**

Applies a function to list/vector of elements. Functions are applied to each element individually.

**lapply**(object, function)

```
# a simple example on list using custom functions

> lapply(list, function(x) str_length(x))
$fruits
[1] 6 5 7 5

$colors
[1] 6 3 6 4
```

# lapply()          list-apply

**lapply() is useful mostly when you have list of data.frames!**

**If you want to recursively work on data.frames in a list, they must contain the same columns**

```
# create a list of two dataframes
> list_df<-list(df_1=df_1, df_2=df_1)

> list_df
$df_1
   fruits colors
1  banana yellow
2   apple    red
3 apricot orange
4   peach   pink

$df_2
   fruits colors
1  banana yellow
2   apple    red
3 apricot orange
4   peach   pink
```

```
df_1<-data.frame(
fruits=fruits, colors=colors)
```

# lapply() list-apply

**lapply() is useful mostly when you have list of data.frames!**

```
# list class of each elements in the list
> lapply(list_df, class)
$df_1
[1] "data.frame"

$df_2
[1] "data.frame"

# count words length in the first column of each element
> lapply(list_df, function(x) str_length(x[,1]))
$df_1
[1] 6 5 7 5

$df_2
[1] 6 5 7 5
```

## lapply()          list-apply

**lapply() is useful mostly when you have list of data.frames!**

```
# select columns in a list of data.frames

> lapply(list_df, "[", c(2))
$df_1
  colors
1 yellow
2    red
3 orange
4   pink

$df_2
  colors
1 yellow
2    red
3 orange
4   pink
```

```
# by column name
> lapply(list_df, "[", "fruits")

# using tidyverse grammar
>lapply(list_df, function(x) x %>%
select(fruits))
```

## sapply() simplified apply

### What it does?

Applies a function to list/vector of elements

**sapply**(object, function)

```
# sapply
# same as lapply but returns a vector
> sapply(list, function(x) str_length(x))
     fruits colors
[1,]      6      6
[2,]      5      3
[3,]      7      6
[4,]      5      4
```

# Map()

## What it does?

Map applies a function to the corresponding elements of given vectors. In some cases acts like **lapply**()

**Map**(function, object)

```
# Map object name to a column

> Map(cbind, list_df, new_col=names(list_df))
$df_1
   fruits colors new_col
1  banana yellow    df_1
2   apple    red    df_1
3 apricot orange    df_1
4   peach   pink    df_1

$df_2
   fruits colors new_col
1  banana yellow    df_2
2   apple    red    df_2
3 apricot orange    df_2
4   peach   pink    df_2
```

```
> lapply(   list_df, function(x) cbind(x,
                new_col=names(list_df)) )
$df_1
   fruits colors new_col
1  banana yellow    df_1
2   apple    red    df_2
3 apricot orange    df_1
4   peach   pink    df_2

$df_2
   fruits colors new_col
1  banana yellow    df_1
2   apple    red    df_2
3 apricot orange    df_1
4   peach   pink    df_2
```

**unwanted behavior!**

## multi-core versions

`library(parallel)` implements parallelized version of many functions

**mclapply**( ..., mc.cores=n )

**mcMap**( ..., mc.cores=n )

...

When are they useful?

- your dataset is really big and your standard function takes too long to run
- when using functions which query a remote database (you can forward multiple queries at once)

# do.call()

**What it does?**

Calls a function on a list of arguments.

**do.call**(function, list)

**lapply**():  function is applied to each element of a list

**do.call**(): function is applied only one time with the arguments provided (list)

```
> do.call(sum, list(1,1))

[1] 2


> do.call(sum, list(c(1,1), c(1,1)))

[1] 4
```

```
> lapply(list(c(1,1), c(1,1)), sum)
[[1]]
[1] 2

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

# do.call() - Collapse lists

**An example use case on lists:**

**You have done your analysis or summarized your data and you need to collapse your list**

**do.call()**

```
> list_df
$df_1
   fruits colors
1  banana yellow
2   apple    red
3 apricot orange
4   peach   pink

$df_2
   fruits colors
1  banana yellow
2   apple    red
3 apricot orange
4   peach   pink
```

```
> do.call(rbind, list_df)

        fruits colors
df_1.1  banana yellow
df_1.2   apple    red
df_1.3 apricot orange
df_1.4   peach   pink
df_2.1  banana yellow
df_2.2   apple    red
df_2.3 apricot orange
df_2.4   peach   pink
```

# rbindlist() - Collapse lists

**An example use case on lists:**

**You have done your analysis or summarized your data and you need to collapse your list**

```
> data.table::rbindlist(list_df, idcol = "df")
```

```
     df  fruits colors
1: df_1  banana yellow
2: df_1   apple    red
3: df_1 apricot orange
4: df_1   peach   pink
5: df_2  banana yellow
6: df_2   apple    red
7: df_2 apricot orange
8: df_2   peach   pink
```

## A real use case example

Dataset from Adamo, Chialva et al., 2020.

The dataset contains CAZymes genes predictions (enzymes involved in the assembly, modification or deconstruction of carbohydrates) in the transcriptomes of 8 fungal species.

- The software who generated predictions outputs 1 file for each species.

###### List all the files in your folder

```
>files<-list.files(pattern = ".txt")
```

## A real use case example

###### Import all the files listed in the object files into a list

- for each elements in the field vector do the read.delim() function
- be careful since text files has an header

## A real use case example

###### Import all the files listed in the object files into a list

- for each elements in the field vector do the read.delim() function
- be careful since text files has an header

```
# import multiple predictions in a list at once
> pred <- lapply(files, function(x) read.delim(x, header = T))
```

## A real use case example

###### Assign the species names to each elements in the list

**Suggestion 1**: the species name is contained in the names of each file in “file” object.

**Suggestion 2**: remove “.txt” file extension using str_replace()

## A real use case example

###### Assign the species names to each elements in the list

**Suggestion 1**: the species name is contained in the names of each file in “file” object.

**Suggestion 2**: remove “.txt” file extension using str_replace()

```
# assign names to list elements
> names(pred)<-str_replace(files, ".txt", "")
```

## A real use case example

###### Filter each data.frame in the list by `X.ofTools` column >=2

- use **filter**()

## A real use case example

###### Filter each data.frame in the list by `X.ofTools` column >=2

- use **filter**()

```
>pred<-lapply(pred, function(x) filter(x, X.ofTools>=2))
```

## A real use case example

###### Add a column in each data.frame indicating the name of the Species

- use **Map**()

## A real use case example

###### Add a column in each data.frame indicating the name of the Species

- use **Map**()

```
>pred<-Map(cbind, pred, SP = names(pred))
```

## A real use case example

###### # operations on columns

# split the Gene.ID column

- use **separate**() to split fields in Gene.ID column
- field separator “|” is a special character: use “\\|” to escape

## A real use case example

###### # operations on columns

# split the Gene.ID column

- use **separate**() to split fields in Gene.ID column
- field separator "|" is a special character: use "\\|" to escape

```
# operations on columns
# split the Gene.ID column
> pred<-lapply(pred, function(x) x %>% separate(Gene.ID,
c("db", "Species", "gene_model", "pred"), remove = F, sep =
"\\|"))
```

## A real use case example

####### Collapse the list into a data.frame

- use **rbindlist**() or **do.call**()

####### export each element of the list into a  .tsv file

- use write.table
- tab separator is “\t”

## A real use case example

####### Collapse the list into a data.frame

- use **rbindlist**() or **do.call**()

```
# collapse data frame
> pred_df<-rbindlist(pred, idcol = "SP_df")
```

####### export each element of the list into a  .tsv file

- use write.table
- tab separator is “\t”

```
> lapply(names(pred), function(x) write.table(pred[x],
file=paste(x, "_export.tsv", sep=""), sep="\t", quote = F))
```