

---

# Lists and Data Frames

---

---

# Lists

R *lists* are objects containing ordered collections of objects (*components*).

The modes of the components of a list can be different (numeric, logic, character, complex).

---

---

# List generation

To generate a list the function `list()` is used:

```
> x <- list(station="Vinadio", elev=1200, month=c("N","D","J"),  
            snowdepth=c(6,21,44))
```

```
> x
```

```
$station
```

```
[1] "Vinadio"
```

```
$elev
```

```
[1] 1200
```

```
$month
```

```
[1] "N" "D" "J"
```

```
$snowdepth
```

```
[1] 6 21 44
```

---

---

An object can be added to a list by passing it as a new component of the list

```
> snowyday=c(4,5,6)
> x[5] <- list(snowyday)
>
> x
$station
[1] "Vinadio"

$elev
[1] 1200

$month
[1] "N" "D" "J"

$snowdepth
[1] 6 21 44

[[5]]
[1] 4 5 6
```

---

---

To join together different lists, say x, y and z, and create a new list xyz, the concatenation function `c()` can be used.

Example:

```
> y<-0
> z <-1
> y<-as.list(y)
> z<-as.list(z)
> xyz<-c(x,X=y,Z=z)
```

```
> xyz
$station
[1] "Vinadio"
$elev
[1] 1200
$month
[1] "N" "D" "J"
$snowdepth
[1] 6 21 44
$snowydays
[1] 4 5 6
$X
[1] 0
$Z
[1] 1
```

---

---

# List indexing

The single element can be accessed using either its position in the list or its name:

Example:

`x[[1]]`

will give "Vinadio"

`x$elev`

will give 1200

N.B.:

The double bracket `[[ ]]` is used to select components of the list

The single bracket `[ ]` is used to select elements of the *i* component

Example:

`x[[4]] [2,3]`

will give: 21,44

---

---

# Data frames

Data frames are objects sharing many of the properties of matrices and of lists:

- they are collections of objects (vectors, matrices, lists or other data frames) of differing types (numeric, logical, character, complex);
  - they are matrix-like structure  
(matrices, lists, data frames provide as many variables to the data frame as they have columns, elements or variables respectively);
  - Vector structures must all have the *same length* (if not they are recycled), matrix structures must all have the *same row size*.
-

---

# Generating a data frame

Objects which satisfies the conditions required can be arranged in a data frame by column using the function **data.frame()**:

```
stationX<- data.frame(date, T, P, p, wind, RH)
```

N.B.: argumenta must have all the same length/number of columns!

A list or a matrix which satisfies the restriction of a data frame can be coerced to a data frame using the functions **as.data.frame()**

---

---

# Accessing variables of a data frame

The components of a data frame can be accessed using:

- the \$ notation:

*name\_dataframe\$name\_component*

- the double brackets `[[ ]]`

*name\_dataframe [[ 1 ]]*

- the functions:

**`attach(name_dataframe)`**

...

**`detach():`**

---

---

Through the function **attach()** it is possible to access the components of a data frame (list) *without quoting the name of the data frame* (list) each time.

The `attach()` function appends the name of the data frame to the searching path.



In this way the components of the data frame become temporarily available as variables under their component names until the **detach()** function is called.

This function detaches the name of the data frame from the searching path, so the components are no longer visible with their names only.

Again they can be accessed only with the list notation:

**stationX\$wind**

---

---

## *ATTENTION*

After the function `attach()` is called, all the operations on the components of the data frame do not affect the original data frame, but a *copy* of its components.

In order to modify the data frame it is necessary to use the list notation (“\$” or `[[ ]]`)

```
dataframe [[1]] <- dataframe [[1]] *(100)
```

---

## Example:

```
>x<-1:3; y<-3:1; z<-0
```

```
>df <- data.frame(x,y,z)
```

```
> df
```

```
  x y z
```

```
1 1 3 0
```

```
2 2 2 0
```

```
3 3 1 0
```

```
> rm(x,y,z)
```

```
> attach(df)
```

```
> z<-x+y
```

```
> detach(df)
```

```
> df
```

```
  x y z
```

```
1 1 3 0
```

```
2 2 2 0
```

```
3 3 1 0
```

```
df$z <-df$x+df$y
```

```
> df
```

```
  x y z
```

```
1 1 3 4
```

```
2 2 2 4
```

```
3 3 1 4
```

---

# Testing and coercing

If you need to verify the *mode* or the *structure* of an object you can use the function **is.()**:

**is.mode(object)**

**is.structure(object)**

- is.integer(x)
- is.double(x)
- is.na(x)
- is.numeric(x)
- is.character(x)
- is.complex(x)
- is.logic(x)
- is.vector(x)
- is.matrix(x)
- is.array(x)
- is.list(x)
- is.data.frame
- is.ts(x)

They return the value TRUE or FALSE

---

---

To coerce an object to have of a specific structure you can use the function

- **as.()** to coerce an object to be the un oggetto ad essere di un tipo specificato

- as.integer(x)

- as.double(x)

- as.na(x)

- as.numeric(x)

- as.character(x)

- as.complex(x)

- as.logic(x)

- as.vector(x)

- as.matrix(x)

- as.array(x)

- as.list(x)

- as.data.frame

- as.ts(x)

---

# Example

```
> a<- 1:5
> a
[1] 1 2 3 4 5
> is.vector(a)
[1] TRUE
> a<-as.matrix(a)
> a
      [,1]
[1,]  1
[2,]  2
[3,]  3
[4,]  4
[5,]  5
> is.matrix(a)
[1] TRUE
> is.integer(a)
[1] TRUE
```