

# Visual R reference card (Draft)

20 janvier 2010

Sylvain Loiseau <sylvain.loiseau@unicaen.fr>  
Université de Caen-Basse Normandie

## 1 Anatomy of a vector

All elements of a vector have a common mode, one of logical, character, and numeric.

"les"	"gli"	"los"
-------	-------	-------

TRUE	FALSE	TRUE
------	-------	------

10	0.32	2
----	------	---

All elements of a vector have an index. They may have a name.

$\text{1}^{\text{st}}$	$\text{2}^{\text{nd}}$	$\text{3}^{\text{rd}}$
1	2	3
"les"	"gli"	"los"

All vectors have two important properties : their mode and their length.

mode (	<table border="1"><tr><td>"les"</td><td>"gli"</td><td>"los"</td></tr></table>	"les"	"gli"	"los"	)	length (	<table border="1"><tr><td>"les"</td><td>"gli"</td><td>"los"</td></tr></table>	"les"	"gli"	"los"	)
"les"	"gli"	"los"									
"les"	"gli"	"los"									
	<table border="1"><tr><td>"character"</td></tr></table>	"character"			<table border="1"><tr><td>3</td></tr></table>	3					
"character"											
3											

mode (	<table border="1"><tr><td>10</td><td>0.32</td><td>2</td></tr></table>	10	0.32	2	)	length (	<table border="1"><tr><td>10</td><td>0.32</td><td>2</td></tr></table>	10	0.32	2	)
10	0.32	2									
10	0.32	2									
	<table border="1"><tr><td>"numeric"</td></tr></table>	"numeric"			<table border="1"><tr><td>3</td></tr></table>	3					
"numeric"											
3											

mode (	<table border="1"><tr><td>TRUE</td><td>FALSE</td><td>TRUE</td></tr></table>	TRUE	FALSE	TRUE	)	length (	<table border="1"><tr><td>TRUE</td><td>FALSE</td><td>TRUE</td></tr></table>	TRUE	FALSE	TRUE	)
TRUE	FALSE	TRUE									
TRUE	FALSE	TRUE									
	<table border="1"><tr><td>"logical"</td></tr></table>	"logical"			<table border="1"><tr><td>3</td></tr></table>	3					
"logical"											
3											

Functions are very precisely defined in terms of mode, number and length of vectors they may take as argument, and mode and length of vector they create.

- length() take one vector of any mode and length and return a numeric vector of length 1.
- mode() take one vector of any mode and length and return a character vector of length 1.

## 2 Creating a vector

```

c ( 1 , 7 , 3 )
  1 7 3

c ( TRUE , FALSE , TRUE )
  TRUE FALSE TRUE

c ( "yes" , "b" , "no" )
  "yes" "b" "no"

5 : 8
  5 6 7 8

seq ( 1 , 4 )
  1 2 3 4

rep ( "yes" , 4 )
  "yes" "yes" "yes" "yes"

rep ( seq ( 1 , 4 ) , 2 )
  1 2 3 4 1 2 3 4

```

The function `c()` take any number of vectors of any length and any mode, but all vectors must have the same mode (see below, "Conversion"). It returns a vector of the same mode as the arguments and whose length is the sum of the length of its arguments.

## 3 Extraction

A vector can be created by extracting some elements of a vector.  
Elements to be extracted can be addressed using their index.

```

10 7 2 [ 2 ]
  7

"les" "gli" "los" [ 1 3 ]
  "les" "los"

```

Index are 1-based. If an index is greater than the number of element in the vector, you get "NA". If the vector has names, their are preserved.

```

TRUE FALSE TRUE [ 1 4 ]
  TRUE NA

10 7 2 [ 1 3 ]
  10 2

```

You can also extract with character or logical vector inside the square brackets of the extraction operator. Elements of a character vector are interpreted as the names of the elements to be extracted (elements must have names!).

```

10 7 2 [ "b" ]
  7

```

Logical vectors must have the same length as the vector to be extracted. Elements are extracted if there is a "TRUE" value at the same position in the logical vector. (If the logical vector is shorter, it is recycled : right (see

below for recycling))

```

10 7 2 [ TRUE FALSE TRUE ] "a" "b" "c" "d" [ TRUE FALSE ]
      10 2                  "a" "c"

```

When extracting, nothing prevent you from reordering elements or extracting several times the same element :

```

"a" "b" "c" "d" [ c ( 1 , 1 , 4 , 2 , 1 ) ]
"a" "b" "c" "d" [ 1 1 4 2 1 ]
"a" "a" "d" "b" "a"

```

## 4 Type conversion

c() coerce arguments to a common mode – all elements of a vector always have a common mode. The character mode always win. Logical always loose.

```

c ( 1 : 3 , FALSE , TRUE )      c ( "oui" , 1 : 3 , FALSE )
c ( 1 2 3 , FALSE , TRUE )      c ( "oui" , 1 2 3 , FALSE )
1 2 3 0 1                       "oui" "1" "2" "3" "FALSE"

```

## 5 Index

Some useful functions give index rather than the actual values.

```

which ( TRUE FALSE FALSE TRUE FALSE TRUE )
      1 4 6

```

```

order ( 10 2 7 15 )      order (
                          

| fr | it | es |
|----|----|----|
| 1  | 2  | 3  |


                          "les" "gli" "los" )
2 3 1 4                  2 1 3

```

```

which.min ( 2 1 3 4 )      which.max ( 2 1 3 4 )
2                          4

```

## 6 Operator

Some numeric operators.

```

5 + 6      5 - 6      5 * 6
11         -1         30

5 / 6      5 < 6      5 >= 6
0.8333333333333333 TRUE FALSE

```

Some logical operators.

`5 == 6`  
`FALSE`

`5 == 5`  
`TRUE`

`TRUE == FALSE`  
`FALSE`

`"oui" == "non"`  
`FALSE`

## 7 Vectorization

Operators – as well as many functions – may operate on vector of any length : the operation is performed on pair of elements of equal index.

`4 3 8 + 32 3 2`  
`36 6 10`

`4 3 8 == 32 3 2`  
`FALSE TRUE FALSE`

## 8 Recycling

In a context where vectorization is allowed, you may provide vectors of unequal length. The shorter is duplicated until its length reach the length of the longer. This is called recycling a vector.

`c ( 1 , 2 , 3 , 4 ) + 1`  
`1 2 3 4 + 1`  
`2 3 4 5`

`5 : 8 > 6`  
`5 6 7 8 > 6`  
`FALSE FALSE TRUE TRUE`

Be careful :

`"..." "yes" "ja" "si" == "ja" "yes"`  
`FALSE TRUE TRUE FALSE`

## 9 Some numerical functions

Some functions for numeric vectors.

`sum ( c ( 2 , 1 , 3 , 4 ) )`  
`sum ( 2 1 3 4 )`  
`10`

`mean ( 2 1 3 4 )`  
`2.5`

`range ( 2 1 3 4 )`  
`1 4`

`rev ( 2 1 3 4 )`  
`4 3 1 2`

`max ( 2 1 3 4 )`  
`4`

`min ( 2 1 3 4 )`  
`1`

```
cumsum ( 2 1 3 4 )
```

```
2 3 6 10
```

TODO : table

## 10 Sorting

Numeric and character vectors can be sorted. Names are preserved.

```
sort ( 2 1 3 4 )      sort ( 2 1 3 4 , decreasing = TRUE )
```

```
1 2 3 4                4 3 2 1
```

```
sort ( "les" "gli" )
```

```
"gli" "les"
```

```
sort (
```

fr	it	es
1	2	3
"les"	"gli"	"los"

```
)
```

it	fr	es
1	2	3
"gli"	"les"	"los"

## 11 Some string functions

```
nchar ( "les" "i" "los" )
```

```
3 1 3
```

Recycling and vectorization are useful with paste(), which concatenates characters string at same index in several characters vector :

```
paste ( "oui" , "non" )      paste ( "oui" , "non" , sep = "" )
```

```
"oui non"                    "ouinon"
```

```
paste ( "oui" "non" , "si" "no" )      paste ( "les" "i" "los" , "oui" )
```

```
"oui si" "non no"                    "les oui" "i oui" "los oui"
```

You can paste more than two vectors of characters :

```
paste ( "(" , names (
```

fr	it	es
1	2	3
"les"	"gli"	"los"

```
), ")" ,
```

fr	it	es
1	2	3
"les"	"gli"	"los"

```
,"les" "gli" "los" , sep = "" )
```

```
paste ( "(" , "fr" "it" "es" , ")" ,
```

fr	it	es
1	2	3
"les"	"gli"	"los"

```
,"les" "gli" "los" , sep = "" )
```

```
"(fr) les" "(it) gli" "(es) los"
```

## 12 Precedence

Operators have precedence (see ?Syntax).

"seq" takes precedence over "+", "seq" takes precedence over logical operators...

1	:	10	+	2
---	---	----	---	---

5	:	8	>	6
---	---	---	---	---

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

+	2
---	---

5	6	7	8
---	---	---	---

>	6
---	---

3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	----	----	----

FALSE	FALSE	TRUE	TRUE
-------	-------	------	------

The order in which the operators are written in the code does not matter!

6	<	5	:	8
---	---	---	---	---

6	<	5	6	7	8
---	---	---	---	---	---

FALSE	FALSE	TRUE	TRUE
-------	-------	------	------

## 13 Factor

TODO

There is numerous situation where values of a vector are seen as modalities, allowing for grouping, etc. (split, rowsum, tapply, etc.)

## 14 Matrix

### 14.1 Creation

```
matrix ( 1 : 6 , nrow = 3 )
matrix ( 1 2 3 4 5 6 , nrow = 3 )
```

1	4
2	5
3	6

```
matrix ( 1 : 6 , 3 )
matrix ( 1 2 3 4 5 6 , 3 )
```

1	4
2	5
3	6

```
matrix ( 1 : 6 , 3 , byrow = TRUE )
matrix ( 1 2 3 4 5 6 , 3 , byrow = TRUE )
```

1	2
3	4
5	6

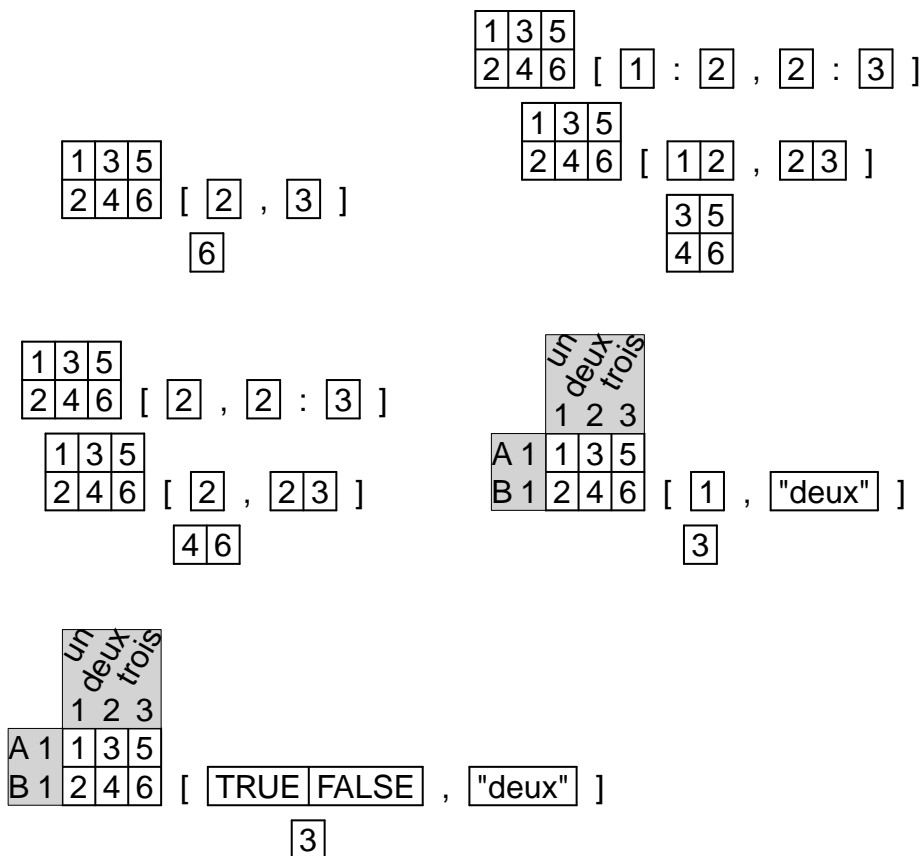
```
matrix ( 1 : 6 , ncol = 3 )
matrix ( 1 2 3 4 5 6 , ncol = 3 )
```

1	3	5
2	4	6

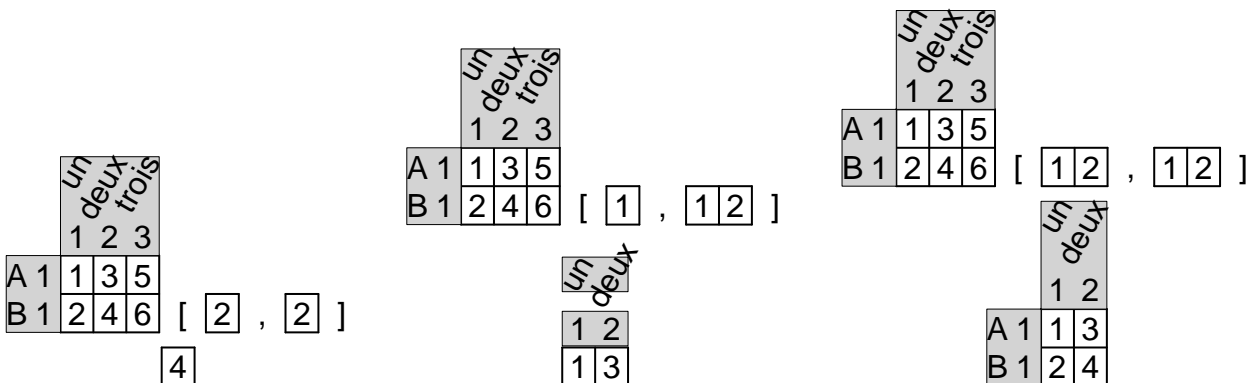
```
matrix ( TRUE , nrow = 2 , ncol = 3 )
```

TRUE	TRUE	TRUE
TRUE	TRUE	TRUE

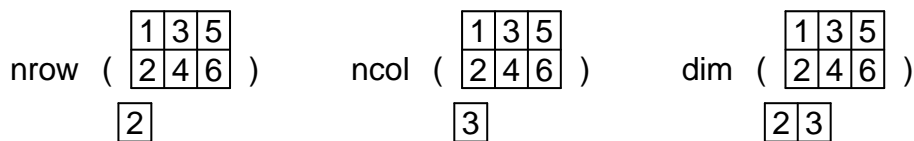
## 14.2 Extraction with a matrix



How does extraction in a matrix preserve names? No name is preserved if you extract a single element; longest dimension's names are preserved if you extract a vector of more than one element, both dimensions are preserved if you extract a sub-matrix:



## 14.3 Properties



`rownames (`

		un	deux	trois
		1	2	3
A	1	1	3	5
B	1	2	4	6

`)`  

"A"	"B"
-----	-----

`colnames (`

		un	deux	trois
		1	2	3
A	1	1	3	5
B	1	2	4	6

`)`  

"un"	"deux"	"trois"
------	--------	---------

## 14.4 Summing a matrix

`sum (`

1	3	5
2	4	6

`)`  

21
----

`rowSums (`

1	3	5
2	4	6

`)`  

9	12
---	----

`colSums (`

1	3	5
2	4	6

`)`  

3	7	11
---	---	----

`rowsum()` perform a `colSums` on each group of rows given by the second argument.

`rowsum (`

1	6
2	7
3	8
4	9
5	10

`,` 

2	1	2	1	3
---	---	---	---	---

`)`  

6	16
4	14
5	10

## 14.5 Changing

`as.vector (`

1	3	5
2	4	6

`)`  

1	2	3	4	5	6
---	---	---	---	---	---

1	3	5
2	4	6

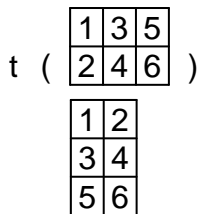
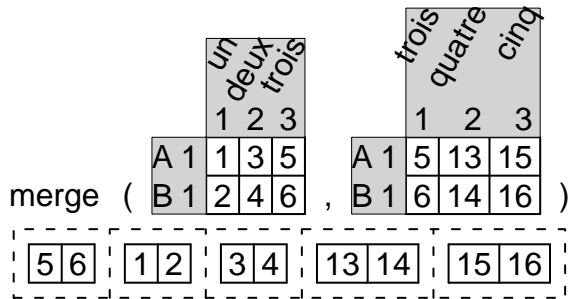
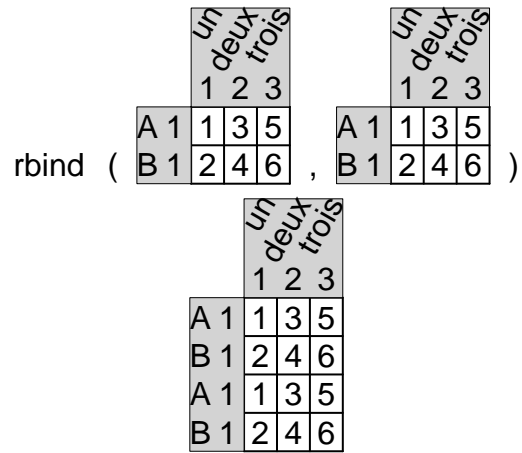
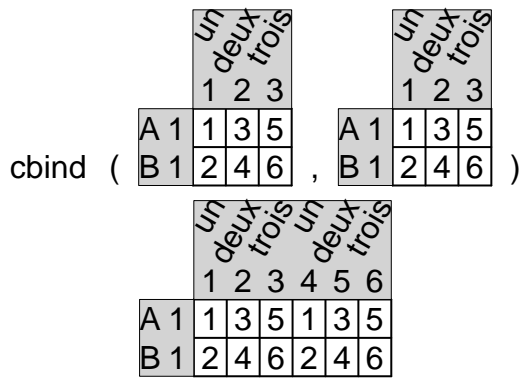
 `+` 

3
---

4	6	8
5	7	9



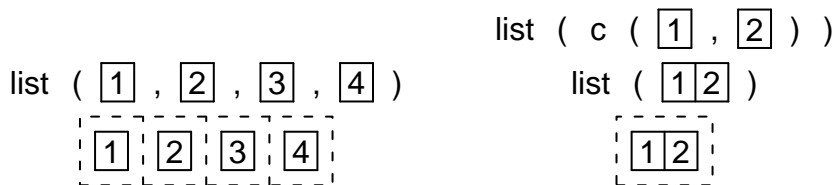


## 15 list

### 15.1 Creation, anatomy

Creating a list by enumerating its components.

A list of length 4 : contains 4 vectors, each of length 1 (left) ; a list of length 1 : contains 1 vector of length 2 (right).

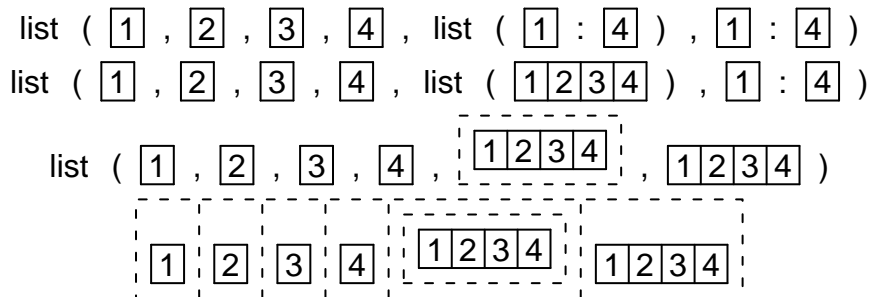


List can contain objects of different mode (left) ; it can contain objects of different dimensions (right).

```
list ( [1] : [3] , matrix ( [1] : [4] , [2] ) , [2] )
list ( [1] : [3] , matrix ( [1 2 3 4] , [2] ) , [2] )
```

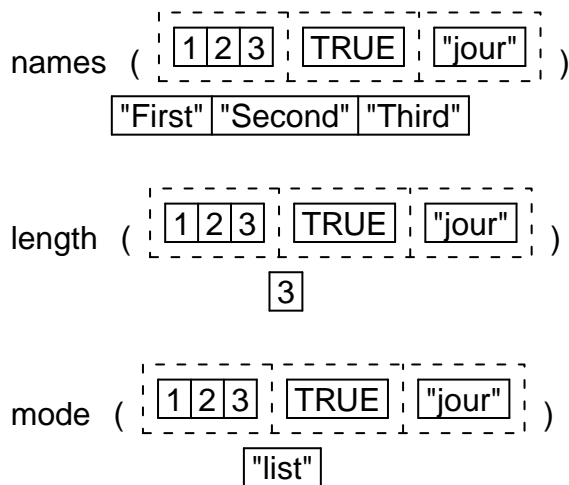


A list can be recursive : a component may be a list.



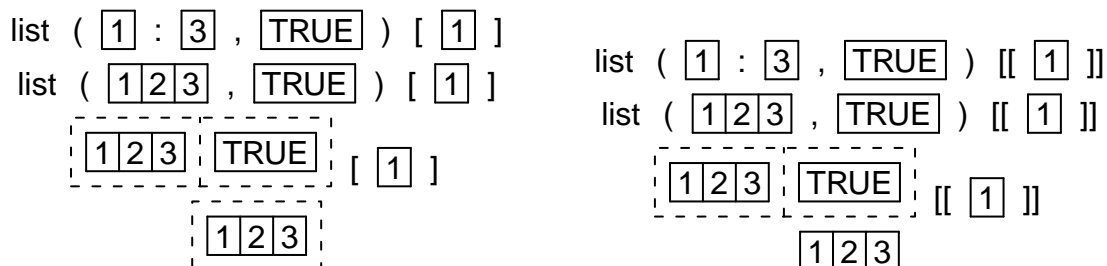
## 15.2 Basic functions

TODO...



## 15.3 Extraction

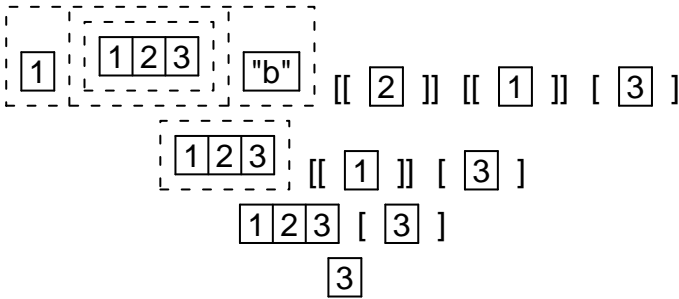
Extracting in a list. The next two figures show the difference between [ and [[ operator on list : the first create a sublist (it extracts elements, exactly as it extract elements from a vector), while the second is completely different : it give the content of one (and only one) element of a list.



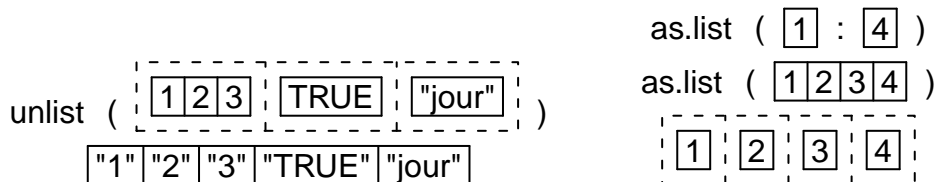
You can use vector of any length within the single-square bracket, while you can address only one element within the double-square-bracket operator, and then use only vector of length 1.

Since a list is a recursive data structure (may contain list), you can use several successive bracket operators in order to go down to the element you're interested in.

With the single-square-bracket operator you cannot walk down though the data structure.

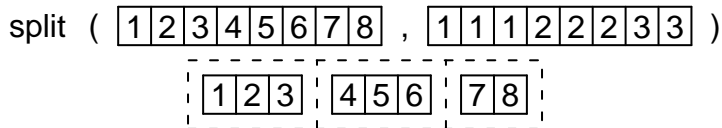


## 15.4 List and vector



## 15.5 List for expressing complex data structure

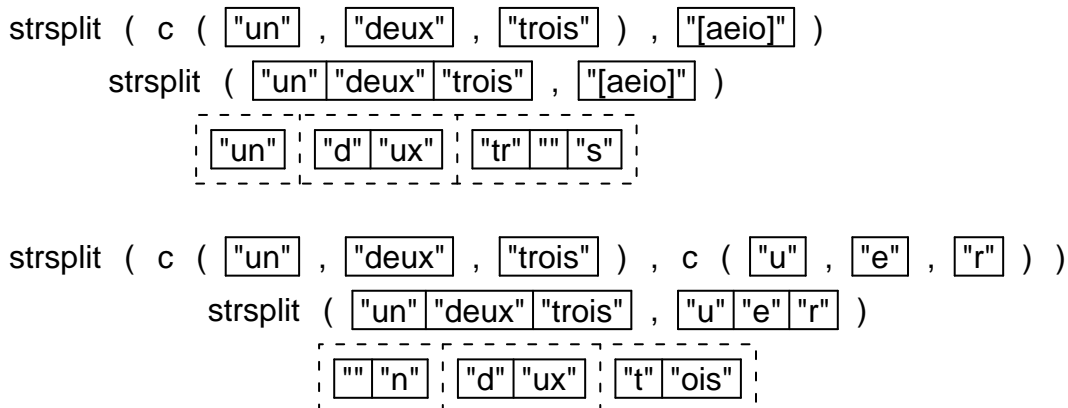
Grouping elements of a vector using a the level of a factors :



See also for instance strsplit() below.

# 16 Regexp

## 16.1 Split strings : strplit()



## 16.2 Extract sub strings : substr()

```
substr ( "trois" , 2 , 3 )  
"ro"
```

```
substr ( c ( "trois" , "quatre" ) , 1 , 3 )  
substr ( "trois" "quatre" , 1 , 3 )  
"tro" "qua"
```

```
substr ( c ( "trois" , "quatre" ) , c ( 2 , 1 ) , c ( 3 , 4 ) )  
substr ( "trois" "quatre" , 2 1 , 3 4 )  
"ro" "quat"
```

## 16.3 Searching elements of a character vector with regexp : grep()

```
grep ( "[dt]" , "un" "deux" "trois" )  
2 3
```

```
grepl ( "[dt]" , "un" "deux" "trois" )  
FALSE TRUE TRUE
```

## 16.4 Substitution : sub()

```
sub ( "[ueaio]" , "v" , "un" "deux" "trois" )  
"vn" "dvux" "trvis"
```

```
gsub ( "[ueaio]" , "v" , "un" "deux" "trois" )  
"vn" "dvvx" "trvvs"
```

## 16.5 Searching substring in elements of character vector : regexpr()

TODO

## 17 Data Frame

TODO