R Programming Language
R01

林　建　甫
C.F. Jeff Lin, MD. PhD.

台北大學統計系助理教授
台北榮民總醫院生物統計顧問
美國密西根大學生物統計博士

R as Objective-Oriented Language

R Basics

- objects
- naming convention
- assignment
- functions
- workspace
- history

Objects

- names
- types of objects: vector, factor, array, matrix, data.frame, ts, list
- attributes
  - mode: numeric, character, complex, logical
  - length: number of elements in object
- creation
  - assign a value
  - create a blank object

Naming Convention

- must start with a letter (A-Z or a-z)
- can contain letters, digits (0-9), and/or periods "."
- case-sensitive
  - `mydata` different from `MyData`
- do not use use underscore "_"

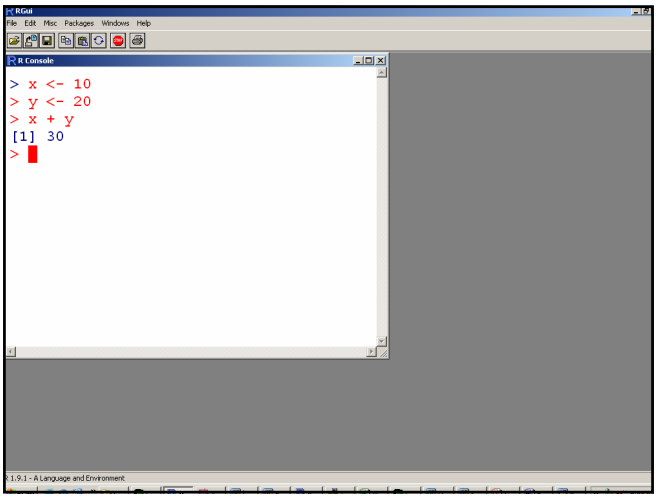
Caution

- Do not use the underscore "_" for names.
- R is case sensitive.
- Try not to use the "=" as it might be confusing and cause problem.
- Remember that if you use "return" command anywhere in your program then it will stop executing at there.

Assignment

- "<-" used to indicate assignment
  - `x<-c(1,2,3,4,5,6,7)`
  - `x<-c(1:7)`
  - `x<-1:4`

- *note: as of version 1.4 "=" is also a valid assignment operator*



Assignment

- Simple operations

- Add: 10 + 20
- Multiply: 10 * 20
- Divide: 10/20
- Raise to a power: 10 ** 20
- Modulo: 10%%20
- Integer division: 10%/%4

Variables and Assignment

```
> a <- 49
> sqrt(a)
[1] 7

> b <- "The dog ate my homework"
> sub("dog","cat",b)
[1] "The cat ate my homework"

> c <- (1+1==3)
> c
[1] FALSE
> as.character(b)
[1] "FALSE"
```

numeric

character string

logical

Assignment

In the previous example x and y are variables. We obtained the sum of x and y by typing

x + y

In the same way we could carry out much more complicated calculations

Generally you can obtain the number (or other value) stored in any letter by typing the letter followed by enter (or by typing print (*letter*) or show (*letter*))

Assignments and Some Basic Math

- The assignment operator consists of '<' and '-' and points from the expression to the name given to that expression
  - x <- 10 assigns the number 10 to the letter x
  - Assignments can be made in either direction
- R as an over-qualified calculator
- Log, exp
- Mean, median, mode, max, min, sd
- Trigonometry
- Set operations
- Logical operators: <, <=, >, >=, ==, !=

R: Logical and Relational Operators

- `==` Equal to
- `!=` Not equal to
- `<` Less than
- `>` Greater than
- `<=` Less than or equal to
- `>=` Greater than or equal to
- `is.na(x)` Missing?
- `&` Logical AND
- `|` Logical OR
- `!` Logical NOT

Basic (Atomic) Data Types

- Logical
  > x <- T; y <- F
  > x; y
  [1] TRUE
  [1] FALSE
- Numerical
  > a <- 5; b <- sqrt(2)
  > a; b
  [1] 5
  [1] 1.414214
- Character
  > a <- "1"; b <- 1
  > a; b
  [1] "1"
  [1] 1
  > a <- "character"
  > b <- "a"; c <- a
  > a; b; c
  [1] "character"
  [1] "a"
  [1] "character"

Getting Stuck at "+" Prompt

- If the "+" prompt continues after hitting return, then enter many ")" to get the ">" prompt
- Then start your expression again

```
> sqrt(
+
+ )))))
Error in parse(text = txt): Syntax error:
        No opening parenthesis, before
        ")" at this point:
sqrt(
))
Dumped
> sqrt(100)
```


Functions

- actions can be performed on objects using functions (note: a function is itself an object)
- have arguments and options, often there are defaults
- provide a result
- parentheses () are used to specify that a function is being called

Missing Values

Variables of each data type (numeric, character, logical) can also take the value NA: not available.
- **NA is not the same as 0**
- **NA is not the same as ""**
- **NA is not the same as FALSE**
- **NA is not the same as NULL**

Operations that involve NA may or may not produce NA:

```
> NA==1
[1] NA
> 1+NA
[1] NA
> max(c(NA, 4, 7))
[1] NA
> max(c(NA, 4, 7), na.rm=T)
[1] 7
```

```
> NA | TRUE
[1] TRUE
> NA & TRUE
[1] NA
```


NA, NaN, and Null

- NA or "Not Available"
  - Applies to many modes – character, numeric, etc.
- NaN or "Not a Number"
  - Applies only to numeric modes
- NULL
  - Lists with zero length

Missing Values

- R is designed to handle statistical data and therefore predestined to deal with missing values
- Numbers that are "not available"

```
> x <- c(1, 2, 3, NA)
> x + 3
[1] 4  5  6 NA
```

- "Not a number"

```
> log(c(0, 1, 2))
[1]      -Inf 0.0000000 0.6931472
> 0/0
[1] NaN
```


R: MissingValues

- Variables of each data type can also take the value NA (for Not Available)
  - NA is not the same as 0
  - NA is not the same as " " (blank, or empty string)
  - NA is not the same as FALSE
- Any computations involving NA *may or may not* produce NA as a result:

```
> 1+NA
[1] NA
> max(c(NA, 4, 7))
[1] NA
> max(c(NA, 4, 7), na.rm=T)
[1] 7
```


Common Object Types for Statistics

Types of Objects

- Vector
- Matrix
- Array
- List
- Factor
- Time series
- Data frame
- Function

typeof() return the type of an R object

Objects

- Mode
  - Atomic mode: logical, numeric, complex or character
  - list, graphics, function, expression, call ..
- Length
  - vector: number of elements
  - matrix, array: product of dimensions
  - list: number of components
  - data frame: number of columns

Objects

- Attributes
  - subordinate names
  - variable names within adata frame
- Class
  - allow for an object oriented style of proramming

Vectors, Matrices, Arrays

- Vector
  - Ordered collection of data of the same data type
  - Example:
    - last names of all students in this class
    - Mean intensities of all genes on an oligonucleotide microarray
  - In R, single number is a vector of length 1
- Matrix
  - Rectangular table of data of the same type
  - Example
    - Mean intensities of all genes measured during a microarray experiment
- Array
  - Higher dimensional matrix

Vectors

- Can think of vectors as being equivalent to a single column of numbers in a spreadsheet.
- You can create a vector using the c( ) function (*concatenate)* as follows:
  - x <- c( )
    - e.g. x <- c(1,2,4,8) creates a column of the numbers 1,2,4,8

Vectors

vector: an ordered collection of data of the same type, or mode

```
> a <- c(1,2,3)
> a*2
[1] 2 4 6
```

Example: the mean spot intensities of all 15488 spots on a microarray is a numeric vector

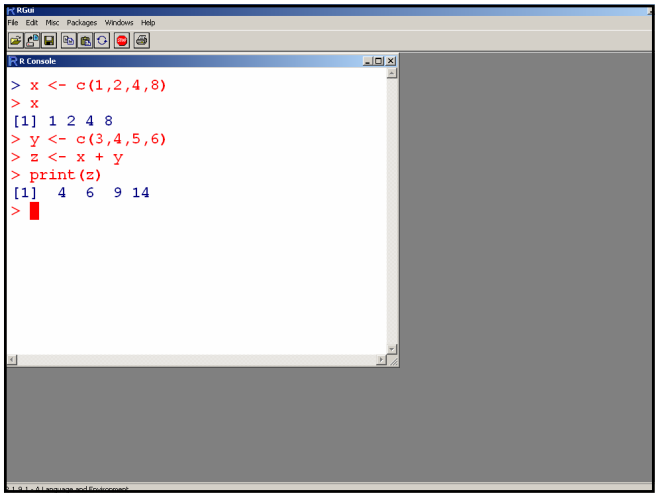
In R, a single number is the special case of a vector with 1 element.

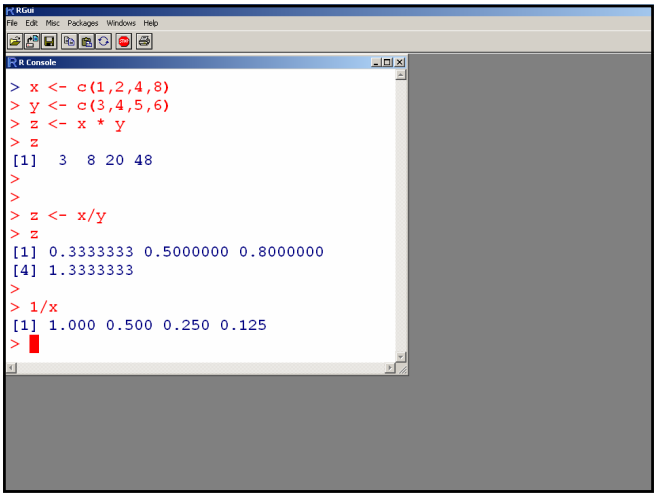
Other vector types: character strings, logical

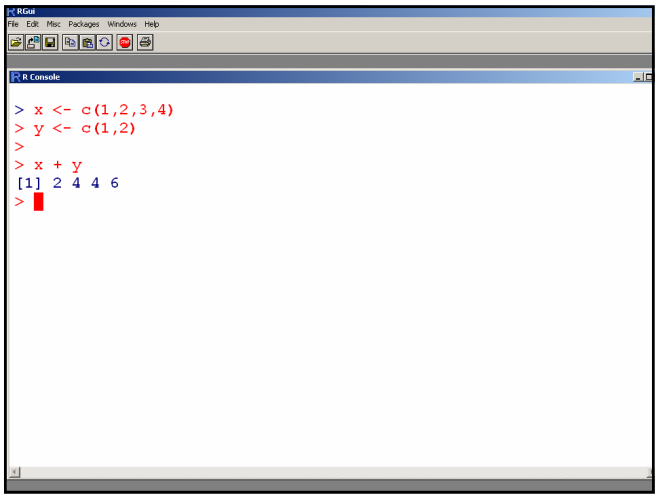
Performing simple operations on vectors

- When you carry out simple operations (+ - * /) on vectors in R that have the same number of entries R just performs the normal operations on the numbers in the vector entry by entry
- If the vectors don't have the same number of entries then R will cycle through the vector with the smaller number of entries
- Vectors have length, but no dimension

  >length(vector)







Vectors

- Vector: Ordered collection of data of the same data type

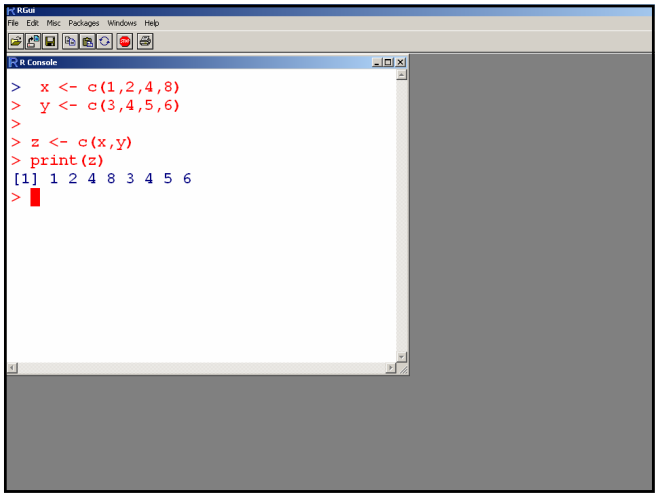
```
> x <- c(5.2, 1.7, 6.3)
> log(x)
[1] 1.6486586 0.5306283 1.8405496
> y <- 1:5
> z <- seq(1, 1.4, by = 0.1)
> y + z
[1] 2.0 3.1 4.2 5.3 6.4
> length(y)
[1] 5
> mean(y + z)
[1] 4.2
```


Vectors

```
> Mydata <- c(2,3.5,-0.2)    # Vector (c="concatenate")
> Colors <- c("Red","Green","Red")    # Character vector

> x1 <- 25:30
> x1
[1] 25 26 27 28 29 30        # Number sequences

> Colors[2]
[1] "Green"                  # One element

> x1[3:5]
[1] 27 28 29                 # Various elements
```


Vectors

vectors (columns of numbers) can be assigned by putting together other vectors



Operation on Vector Elements

```
> Mydata
[1] 2 3.5 -0.2
```

```
> Mydata > 0
[1] TRUE TRUE FALSE
```

- Test on the elements

```
> Mydata[Mydata>0]
[1] 2 3.5
```

- Extract the positive elements

```
> Mydata[-c(1,3)]
[1] 3.5
```

- Remove elements

Vector Operations

```
> x <- c(5,-2,3,-7)
> y <- c(1,2,3,4)*10        Operation on all the elements
> y
[1] 10 20 30 40

> sort(x)                   Sorting a vector
[1] -7 -2 3 5

> order(x)
[1] 4 2 3 1                 Element order for sorting

> y[order(x)]
[1] 40 20 30 10             Operation on all the components

> rev(x)                    Reverse a vector
[1] -7 3 -2 5
```


c() & rev()

- c() concatenates or combines numbers inside () into a vector or list

```
> c(1,3,5,7)
[1] 1 3 5 7

> rev(c(1,3,5,7))
[1] 7 5 3 1
```


length(), mode() & names()

```
> x<-c(1,3,5,7)
> length(x)
[1] 4
> mode(x)
[1] "numeric"
> names(x)
NULL
```


seq()

- seq() generates sequence
  - Can specify length of sequence and increment

```
> seq(1:5)
[1] 1 2 3 4 5
> seq(5,1,by=-1)
[1] 5 4 3 2 1
> seq(5)
[1] 1 2 3 4 5
```


seq()

```
> 1.1:5
[1] 1.1 2.1 3.1 4.1
> 4:-5
 [1]  4  3  2  1  0 -1 -2 -3 -4 -5
> seq(-1,2,0.5)
[1] -1.0 -0.5  0.0  0.5  1.0  1.5  2.0
> seq(1,by=0.5,length=5)
[1] 1.0 1.5 2.0 2.5 3.0
```


rep()

- rep() replicates elements

```
> rep(1,5)
[1] 1 1 1 1 1
> rep(1:2,3)
[1] 1 2 1 2 1 2
> rep(1:2,each=3)
[1] 1 1 1 2 2 2
> rep(1:2,each=3,len=4)
[1] 1 1 1 2
> rep(1:2,each=3,len=7)
[1] 1 1 1 2 2 2 1
> rep(1:2,each=3,time=2)
 [1] 1 1 1 2 2 2 1 1 1 2 2 2
```

sort() & rank()

- sort(vector) and rank(vector) sorts items in and rank vector

```
> x<-c(8,6,9,7)
> sort(x)
[1] 6 7 8 9
> rank(x)
[1] 3 1 4 2
> rank(x)[1]
[1] 3
> x[rank(x) = =1]
[1] 6
> x[rank(x)]
[1] 9 8 7 6
```


rank() & order()

```
> x<-c(8,6,9,7)
> order(x)
[1] 2 4 1 3
> rank(x)
[1] 3 1 4 2

> x[order(x)]
[1] 6 7 8 9
> x[rank(x)]
[1] 9 8 7 6
```


Matrices and Arrays

matrix: rectangular table of data of the same type

Example: the expression values for 10000 genes for 30 tissue biopsies is a numeric matrix with 10000 rows and 30 columns.

array: 3-,4-,..dimensional matrix

Example: the red and green foreground and background values for 20000 spots on 120 arrays is a 4 x 20000 x 120 (3D) array.

Matrix

- a matrix is a vector with an additional attribute (dim) that defines the number of columns and rows
- only one mode (numeric, character, complex, or logical) allowed
- can be created using `matrix()`

```
x<-matrix(data=0,nr=2,nc=2)
        or
x<-matrix(0,2,2)
```


Matrices

- Matrices have length and dimensions
  - >length(M); dim(M)
- Generating matrices
  - By combining vectors: rbind(); cbind()
  - >matrix() command
- Transpose
  - >t(M)
- Diagonal
  - >diag(M)
- Inverse
  - >solve(M)
- Multiplying matrices: M%*%N
- Indexing matrices

Matrices

- Matrix: Rectangular table of data of the same type

```
> m <- matrix(1:12, 4, byrow = T); m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

Matrices

```
> y <- -1:2
> m.new <- m + y
> t(m.new)
     [,1] [,2] [,3] [,4]
[1,]    0    4    8   12
[2,]    1    5    9   13
[3,]    2    6   10   14
> dim(m)
[1] 4 3
> dim(t(m.new))
[1] 3 4
```


Matrices

Matrix: Rectangular table of data of the same type

```
> x <- c(3,-1,2,0,-3,6)
> x.mat <- matrix(x,ncol=2)        # Matrix with 2 cols
> x.mat
     [,1] [,2]
[1,]    3    0
[2,]   -1   -3
[3,]    2    6
> x.mat <- matrix(x,ncol=2, byrow=T) # By row creation
> x.mat
     [,1] [,2]
[1,]    3   -1
[2,]    2    0
[3,]   -3    6
```


Dealing with Matrices

```
> x.mat[,2]   # 2nd col
[1] -1 0 6

> x.mat[c(1,3),]   #1st and 3rd lines
     [,1] [,2]
[1,]    3   -1
[2,]   -3    6

> x.mat[-2,]   #No 2nd line
     [,1] [,2]
[1,]    3   -1
[2,]   -3    6
```


Dealing with Matrices

```
> dim(x.mat)  # Dimension
[1] 3 2
> t(x.mat)       # Transpose
     [,1] [,2] [,3]
[1,]    3    2   -3
[2,]   -1    0    6
> x.mat %*% t(x.mat)   # %*% Multiplication (inner product)
     [,1] [,2] [,3]
[1,]   10    6  -15
[2,]    6    4   -6
[3,]  -15   -6    4
> solve()      # Inverse of a square matrix
> eigen()      # Eigenvectors and eigenvalues
```


Subsetting

- It is often necessary to extract a subset of a vector or matrix
- R offers a couple of neat ways to do that

```
> x <- c("a", "b", "c", "d", "e", "f", "g", "h")
> x[1]
> x[3:5]
> x[-(3:5)]
> x[c(T, F, T, F, T, F, T, F)]
> x[x <= "d"]
> m[,2]
> m[3,]
```


Generate a Matrix

```
> xmat<-matrix(1:12,nrow=3,byrow=T)
> xmat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> length(xmat)
[1] 12
> dim(xmat)
[1] 3 4
> mode(xmat)
[1] "numeric"
> names(xmat)
NULL
> dimnames(xmat)
NULL
```

Generate a Matrix

```
> dimnames(xmat)<-list(c("A","B","C"), c("W","X","Y","Z"))

> dimnames(xmat)
[[1]]
[1] "A" "B" "C"
[[2]]
[1] "W" "X" "Y" "Z"

> xmat
   W  X  Y  Z
A  1  2  3  4
B  5  6  7  8
C  9 10 11 12
```


Generate a Matrix

```
> matrix(0,3,3)
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```


Diagonal Element of a Matrix

```
> m <- matrix(1:12, 4, byrow = T)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> diag(m)
[1] 1 5 9
```


Diagonal Element of a Matrix

```
> diag(k)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```


Inverse of Matrices

```
> m<-matrix(c(1,3,5,,9,11,13,15,19,21),3,byrow=T)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    9   11   13
[3,]   15   19   21
> solve(m)
       [,1]    [,2] [,3]
[1,] -0.5000  1.0000 -0.5
[2,]  0.1875 -1.6875  1.0
[3,]  0.1875  0.8125 -0.5
```


rbind() & cbind()

```
> x<-c(1,2,3)
> y<-matrix(0,3,3)

> rbind(y,x)
  [,1] [,2] [,3]
     0    0    0
     0    0    0
     0    0    0
x    1    2    3

> cbind(y,x)
          x
[1,] 0 0 0 1
[2,] 0 0 0 2
[3,] 0 0 0 3
```

Multiplication

```
> x<-matrix(1:4,2,byrow=T)
> y<-matrix(1:4,2,byrow=T)
> x*y          # element wise
     [,1] [,2]
[1,]    1    4
[2,]    9   16

> x%*%y    #
     [,1] [,2]
[1,]    7   10
[2,]   15   22
```



```
> x%o%x
, , 1, 1
     [,1] [,2]
[1,]    1    2
[2,]    3    4

, , 2, 1
     [,1] [,2]
[1,]    3    6
[2,]    9   12

, , 1, 2
     [,1] [,2]
[1,]    2    4
[2,]    6    8

, , 2, 2
     [,1] [,2]
[1,]    4    8
[2,]   12   16
```


Array

Arrays are generalized matrices by extending the function **dim()** to mor thantwo dimensions.

```
> xarr<-array(c(1:8,11:18,111:118),dim=c(2,4,3)) # row, col, array
> xarr
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8

, , 2
     [,1] [,2] [,3] [,4]
[1,]   11   13   15   17
[2,]   12   14   16   18

, , 3
     [,1] [,2] [,3] [,4]
[1,]  111  113  115  117
[2,]  112  114  116  118
```


Lists, Factors and Data Frames

Lists

list: ordered collection of data of arbitrary types.

Example:

```
> doe <- list(name="john",age=28,married=F)
> doe$name
[1] "john"
> doe$age
[1] 28
> doe[[3]]
[1] FALSE
```

Typically, vector elements are accessed by their index (an integer) and list elements by $name (a character string). But both types support both access methods. Slots are accessed by @name.

Lists

vector: an ordered collection of data of the same type.

```
> a = c(7,5,1)
> a[2]
[1] 5
```

list: an ordered collection of data of arbitrary types.

```
> doe = list(name="john",age=28,married=F)
> doe$name
[1] "john"
> doe$age
[1] 28
```

Typically, vector elements are accessed by their index (an integer), list elements by their name (a character string). But both types support both access methods.

Lists

- A list is an object consisting of objects called components.
- The components of a list don't need to be of the same mode or type and they can be a numeric vector, a logical value and a function and so on.
- A component of a list can be referred as aa[[l]] or aa$times, where aa is the name of the list and times is a name of a component of aa.

Lists

- The names of components may be abbreviated down to the minimum number of letters needed to identify them uniquely.
- aa[[1]] is the first component of aa, while aa[1] is the sublist consisting of the first component of aa only.
- There are functions whose return value is a List. We have seen some of them, eigen, svd, …

Lists Are Very Flexible

```
> my.list <- list(c(5,4,-1),c("X1","X2","X3"))
> my.list
[[1]]:
[1]  5  4 -1

[[2]]:
[1] "X1" "X2" "X3"

> my.list[[1]]
[1]  5  4 -1

> my.list <- list(c1=c(5,4,-1),c2=c("X1","X2","X3"))
> my.list$c2[2:3]
[1] "X2" "X3"
```


Lists: Session

```
Empl <- list(employee="Anna", spouse="Fred",
    children=3, child.ages=c(4,7,9))
Empl[[4]]
Empl$child.a
Empl[4]   # a sublist consisting of the 4th component of Empl
names(Empl) <- letters[1:4]
Empl <- c(Empl, service=8)
unlist(Empl)  # converts it to a vector. Mixed types will be converted
    to character, giving a character vector.
```


More Lists

```
> x.mat
     [,1] [,2]
[1,]    3   -1
[2,]    2    0
[3,]   -3    6

> dimnames(x.mat) <- list(c("L1","L2","L3"), c("R1","R2"))

> x.mat
    R1 R2
L1  3 -1
L2  2  0
L3 -3  6
```


Factor and factor()

A character string can contain arbitrary text. Sometimes it is useful to use a limited vocabulary, with a small number of allowed words. A factor is a variable that can only take such a limited number of values, which are called levels.

```
> gender<-c("male","female",
    "male","male","female","female")
> gender
[1] "male"   "female" "male"   "male"   "female" "female"
> factor(gender)
[1] male   female male   male   female female
Levels: female male
```

factor() and levels()

```
intensity<-
    factor(c("Hi","Med","Lo","Hi","Lo","Med",
    "Lo","Hi","Med"))
> intensity
[1] Hi  Med Lo  Hi  Lo  Med Lo  Hi  Med
Levels: Hi Lo Med
```


factor() and levels()

```
> intensity<-
    factor(c("Hi","Med","Lo","Hi","Lo","Med",
   "Lo","Hi","Med"), levels=c("Hi","Med","Lo"))
> intensity
[1] Hi  Med Lo  Hi  Lo  Med Lo  Hi  Med
Levels: Hi Med Lo
```


factor() and levels()

```
intensity<-
    factor(c("Hi","Med","Lo","Hi","Lo","Med",
    "Lo","Hi","Med"), levels=c("Hi","Med","Lo"),
    labels=c("HiDOse","MedDOse","LoDose"))
> intensity
[1] HiDOse  MedDOse LoDose  HiDOse  LoDose
    MedDOse LoDose  HiDOse  MedDOse
Levels: HiDOse MedDOse LoDose
```


factor(), ordered() and levels()

```
intensity<-
    ordered(c("Hi","Med","Lo","Hi","Lo","Med",
    "Lo","Hi","Med"))
> intensity
[1] Hi  Med Lo  Hi  Lo  Med Lo  Hi  Med
Levels: Hi < Lo < Med
```

Oooooop! This is not what you want!

factor(), ordered() and levels()

```
intensity<-
    ordered(c("Hi","Med","Lo","Hi","Lo","Med",
   "Lo","Hi","Med"), levels=c("Lo","Med", "Hi"))
> intensity
[1] Hi  Med Lo  Hi  Lo  Med Lo  Hi  Med
Levels: Lo < Med < Hi
```

Ordinal Variable!

Data Frames

data frame: represents a spreadsheet.
Rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.
...

Data Frames

```
# R data ToothGrowth
# The Effect of Vit. C on Tooth Growth in Guinea Pigs
> ToothGrowth
    len supp dose
1   4.2   VC  0.5
2  11.5   VC  0.5
3   7.3   VC  0.5
4   5.8   VC  0.5
……
58 27.3   OJ  2.0
59 29.4   OJ  2.0
60 23.0   OJ  2.0
```


Data Frames

- A data frame is a list with class "data.frame". There are restrictions on lists that may be made into data frames.
- a. The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or other data frames.
- b. Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.

Data Frames

- c. Numeric vectors and factors are included as is, and non-numeric vectors are coerced to be factors, whose levels are the unique values appearing in the vector.
- d. Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

Data Frame

- several modes allowed within a single data frame
- can be created using `data.frame()`

```
L<-LETTERS[1:4] #A B C D
x<-1:4          #1 2 3 4
data.frame(x,L) #create data frame
```

- `attach()` and `detach()`
  - the database is attached to the R search path so that the database is searched by R when it is evaluating a variable.
  - objects in the database can be accessed by simply giving their names

Data Elements

- select only one element
  - `x[2]`
- select range of elements
  - `x[1:3]`
- select all but one element
  - `x[-3]`
- slicing: including only part of the object
  - `x[c(1,2,5)]`
- select elements based on logical operator
  - `x(x>3)`

Subsetting

Individual elements of a vector, matrix, array or data frame are accessed with "[ ]" by specifying their index, or their name

```
> ToothGrowth[1:3,]
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5

> ToothGrowth[1:2,1:2]
   len  supp
1  4.2   VC
2 11.5   VC
```

Labels in Data Frames

```
> labels(ToothGrowth)
[[1]]
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
[13] "13" "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24"
[25] "25" "26" "27" "28" "29" "30" "31" "32" "33" "34" "35" "36"
[37] "37" "38" "39" "40" "41" "42" "43" "44" "45" "46" "47" "48"
[49] "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"

[[2]]
[1] "len"  "supp" "dose"
```


Finding out about a data object

mode ( ): tells you the storage 'mode' of an object (i.e. whether it is a numeric vector, or a list etc.)

attributes ( ): provides information about the data object

class( ): provides informaiton about the object's class. The class of an object often determines how the data object is handled by a function.

You can also set the object's mode, attributes or class using the above functions.

e.g. mode (x) <- "numeric"

What type is my data?

`class`	Class from which object inherits (vector, matrix, function, logical, list, …)
`mode`	Numeric, character, logical, …
`storage.mode` `typeof`	Mode used by R to store object (double, integer, character, logical, …)
`is.function`	Logical (TRUE if function)
`is.na`	Logical (TRUE if missing)
`names`	Names associated with object
`dimnames`	Names for each dim of array
`slotNames`	Names of slots of BioC objects
`attributes`	Names, class, etc.

Data Import & Entry

Topics

- Datasets that come with R
- Inputting data from a file
- Writing data to a file
- Writing data to the clipboard
- Exchanging data between programs
- NB: saving the workspace

R comes with several pre-packaged datasets

You can access these datasets with the *data* function

data ( ) *gets you a list of all the datasets*

data (Titanic) *loads a dataset about passengers on the Titanic (for example)*

summary (Titanic) *provides some summary information about the dataset Titanic*

attributes(Titanic) *provides some more information*

Typing the dataset name on its own (followed by Enter*) will display the data*

Data

>summary(data)
>names(data)
>attributes(data)

Editing data

>fix(data) or >edit(data)

>data$var

>attach(data) # in order to remove need of '$'
>detach(data)

Data Entry & Editing

- start editor and save changes
  - `data.entry(x)`
- start editor, changes not saved
  - `de(x)`
- start text editor
  - `edit(x)`

The *attach* and *detach* functions

The *attach* function makes all the objects in a list or data frame accessible from outside the list or data frame. E.g. instead of typing my_list$age to access the vector 'age' in the list my_list you can just type 'age' (provided there is no other vector called 'age' in the main workspace).

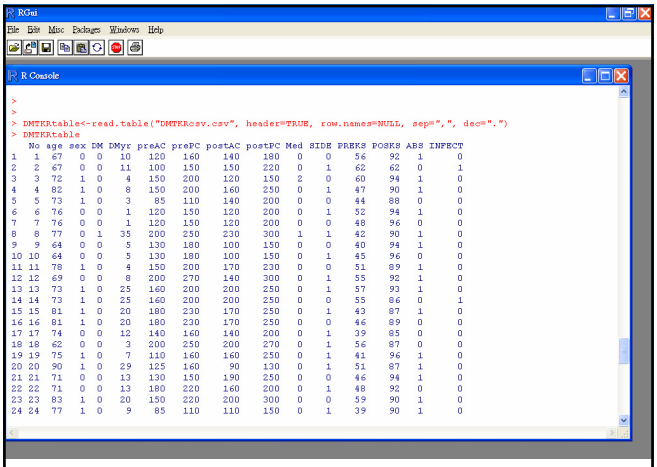
The *detach* function undoes this

Importing Data

- `read.table()`
  - reads in data from an external file
- `data.entry()`
  - create object first, then enter data
- `c()`
  - concatenate
- `scan()`
  - prompted data entry
- R has ODBC for connecting to other programs

Importing Data

```
> # Data Managements
> setwd("C://temp//Rdata")
> DMTKRtable<-read.table("DMTKRcsv.csv",
    header=TRUE, row.names=NULL, sep=",", dec=".")
> DMTKRtable
```

Importing Data

```
> setwd("C://temp//Rdata")
> DMTKRcsv<-read.csv("DMTKRcsv.csv",
      header = TRUE, sep = ",", dec=".")
> DMTKRcsv
> attach(DMTKRcsv)

> scan(file = "DMTKRcsv.csv", skip=1, sep = ",",
      dec = ".")
```


Loading

- Stata, SPSS, SAS files
  - Library(foreign)
    - Stata: read.dta
    - SPSS: read.spss
    - SAS: read.xport (must first create export file in SAS)
- Excel files
  - Files must be saved as comma separated value or .csv
  - read.table, read.csv, read.csv2: identical except for defaults
- Watch the direction of '/'!
- >load(".Rdata")
- Loading and running R programs
  - >source(".R")

Writing data to a file (the *write* and *write.table* functions)

Change directory on the file menu then

write (q, file = "filename", ncol = 2)

(for vector, ncol specifies the number of columns in output)

write.table (q, file = "filename")

(works quite well for a data frame)

as always there are many optional arguments

Exporting Data

```
> #write data out
> cat("2 3 5 7", "11 13 17 19", file="ex.dat", sep="\n")
# Read in ex.dat again
> scan(file="ex.dat", what=list(x=0, y="", z=0),
      flush=TRUE)


df <- data.frame(a = I("a \" quote"))
write.table(df)
write.table(df, qmethod = "double")
write.table(df, quote = FALSE, sep = ",")
```