Course_2_Week1_Lecture_Notes

Zhenzhuo A. C.

26.03.22

Two Basic Operators <- and :

1. <- is called the "assignment operator", which assigns a value to a symbol

- **example**:

```
x<-1
x
```

```
## [1] 1
```

The created symbol is called x, and the assigned value is 1. "x is 1" is a R expression.

At the same time, x is a numeric object with 1 element.

- **another example**:

```
msg <- "hello"
msg
```

```
## [1] "hello"
```

msg is also a symbol assigned to the string *hello*, making msg a character vector.

- **special case with ##**

```
```{r}
hello <- ## I am still thinking
hello
```
```

```
Error: object 'hello' not found
```

Everything on the right side of double hash is comments. In the above expression, no value has been assigned to the symbol *hello*. Therefore, there is no print-out result.

2. : is the operator for integer sequences creation.

```
x <- 1:20
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Data Types in R

Atomic classes

R has 5 basic "atomic" classes of objects:
1. character
2. numeric (real numbers)
3. integer
4. complex
5. logical (TRUE/FALSE)

Data Type: Vectors

The most basic factor in R is a vector that **only** contains **objects of the same class**

Vectors can be created using vector() function. E.g. An numeric vector with length = 10 by default.

```
x<-vector("numeric", length = 10)
x
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

The c() function can also be used to create vectors of objects.

```
x<- c(0.5, 0.6)
class(x)
```

```
## [1] "numeric"
```

```
z<- c(TRUE, FALSE) ##logical
class(z)
```

```
## [1] "logical"
```

```
u <- c(T, F)
class(u)
```

```
## [1] "logical"
```

```
v<- c("a","b","c")
class(v)
```

```
## [1] "character"
```

```
w<- 9:20
class(w)
```

```
## [1] "integer"
```

```
k<-c(1+0i, 2+4i)
class(k)
```

```
## [1] "complex"
```

Data Type: Lists

However, if a “vector” contains objects of different classes, then it is a *list*. The class of list follows the rule of **least common denominator**.

```
m<- c(1.7, "a")
class(m)
```

```
## [1] "character"
```

```
n<-c(TRUE,2)
class(n)
```

```
## [1] "numeric"
```

The reason why number 2 is the least denominator because TRUE and FALSE these 2 logical objects symbolize number 1 and 0.

```
p<-c("a", TRUE)
class(p)
```

```
## [1] "character"
```

The output of a list separate every element because they are not of a same class.

```
q<-list(1, "a", TRUE, 1+4i)
q
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

Data Type: Number

Numbers are generally numeric objects. NaN represents undefined value.

Data Type: Attributes

Attributes can also be objects in R.

1. names, dimnames (i.e. dimension names)
2. dimensions (e.g. matrices: number of columns and rows symbolizes dimensions; multidimensional arrays)
3. class
4. length
5. other user-defined attributes/metadata (what you can define separately for an object using various attribute functions)

- Attributes can be accessed using attributes() function.
- Every object has a class and length. For vectors it is very obvious: the length of the object equals the number of elements in the vector.

Explicit Coercion between Classes in Vector

Using the as.numeric(), as.logical(), as.character() function, respectively, to convert the vector to your wished class.

```
r<-0:6
r
```

```
## [1] 0 1 2 3 4 5 6
```

```
class(r)
```

```
## [1] "integer"
```

```
as.logical(r)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
as.character(r)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

However, coercion does not work always:

```
t<-c("a","b","c")
as.numeric(t)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA
```

```
as.logical(t)
```

```
## [1] NA NA NA
```

```
as.complex(t)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA
```

if the coercion does not work, the output will be NA.

Data Type: Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2, namely (nrow, ncol).
- You can use matrix() function to create a matrix.
- dim() function to check the dimension.
- attribute() function to convert the dimension to a vector.

```
a <- matrix(nrow = 2, ncol = 3)
a
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
dim(a)
```

```
## [1] 2 3
```

```
attributes(a)
```

```
## $dim
## [1] 2 3
```

When assigning the values, matrices are constructed *column-wise*

```
b<-matrix(1:6, nrow = 2, ncol = 3)
b
```

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

And we can also use dim() to convert a vector to a matrix.

```
e<-1:10
dim(e) = c(2,5)
e
```

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

Another way to create matrices is to use column binding or row binding.

```
e<-1:10
f<-2:11
cbind(e,f)
```

```
##        e  f
##  [1,]  1  2
##  [2,]  2  3
##  [3,]  3  4
##  [4,]  4  5
##  [5,]  5  6
##  [6,]  6  7
##  [7,]  7  8
##  [8,]  8  9
##  [9,]  9 10
## [10,] 10 11
```

```
rbind(e,f)
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## e    1    2    3    4    5    6    7    8    9    10
## f    2    3    4    5    6    7    8    9   10    11
```

Data Type: Factors

Factors are used to represent categorical data, which can be ordered or unordered. 1. integer factor where each integer has a label (e.g. 1 represents "high, 2 represents"medium", 3 represents "low") 2. professor, associated professor, visiting professor… (categorical but ordered)

```
g <- factor(c("no", "no", "yes","no","yes"))
g
```

```
## [1] no  no  yes no  yes
## Levels: no yes
```

The two levels of this factor are "no" and "yes".

Use table() function to count the frequency of these two levels:

```
table(g)
```

```
## g
##  no yes
##   3   2
```

The unclass() function strips out the class of a factor:

```
unclass(g)
```

```
## [1] 1 1 2 1 2
## attr(,"levels")
## [1] "no"  "yes"
```

The order of levels can be set using *levels* argument to a factor. This is especially useful in linear modelling because the first level is usually used as the baseline level.

```
g <- factor(c("no", "no", "yes","no","yes"), levels=c("yes", "no"))
g
```

```
## [1] no  no  yes no  yes
## Levels: yes no
```

Data Type: Missing Values

Missing values are denoted by NA or NaN. NaN stands for undefined mathematical operations, while NA stands generally all missing values, including NaN.

```
h<-c(1,2,4,NA,5,7)
is.na(h)
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE
```

```
is.nan(h)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
i<-c(1,2,4,NA,NaN,5,7)
is.na(i)
```

```
## [1] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
```

```
is.nan(i)
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Data Type: Data Frames

Data frames are used to store tabular data.
- Data frames are represented as a special type of list where every element of the list has to have the same length.
- Each element of the list can be thought as a column and the length of each element of the list is the number of rows.
- Unlike matrices, data frames can store different classes of objects in each column like lists.
- Every row of a data frame has a name, i.e. a special attribute called *row.names*.(e.g. each row can represent a subject of study, and the row name can be the subject ID)
- Data frames are usually created by calling *read.csv()* or *read.table()*.
- Can be converted to a matrix using data.matrix().

```
l<-data.frame(food_id = 1:4, food_name= c("pasta", "rice","bread", "couscous"))
l
```

food_id <int>	food_name <chr>
1	pasta
2	rice
3	bread
4	couscous

4 rows

Data Type: Names

```
names(l)
```

```
## [1] "food_id"   "food_name"
```

Lists can also have names. a, b, c are the names in the example below, and 1, 2, 3 are the associated values.

```
o<-list(a=1, b=2, c=3)
o
```

```
## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3
```

We can assign names to matrices using dimname() function.

```
w<-matrix(1:4, nrow = 2, ncol = 2)
dimnames(w)<- list(c("a","b"), c("c", "d"))
w
```

```
##   c d
## a 1 3
## b 2 4
```

> Data Type Summary
> 1. atomic classes: numeric, logical, character, integer, complex
> 2. vectors, lists
> 3. factors
> 4. missing values
> 5. data frames
> 6. names

Functions for Data Reading in R

read.csv(), read.table() for tabular data

These functions read text files that contain data stored in rows and columns type of format and return a data frame in R.

read.table() has some important *arguments*
- *file*, the name of a file or connection
- *header*, logical indicating if the file has a header line
- *sep*, a string indicating how the columns are separated
- *colClasses*, a character vector indicating the class of each column in the dataset
- *nrows*, the number of rows in the dataset
- *comment.char*, a character string indicating the comment character
- *skip*, the number of lines to skip from the beginning
- *stringAsFactors*, should character variables be coded as factors?

read.csv() is identical to read.table specified for comma as defaulted separator.

readLines()

Use this function to read lines of a text file. This can be any kind of files, just like a vector of characters in R.

source()

For reading in R code files (inverse of dump()).

dget()

For reading in R code files (inverse of dput()).

load()

For reading in saved workspaces.

unserialize()

For reading single R objects in binary form.

Functions for Data Writing in R

- write.table
- writeLines
- dump
- dput
- save
- serialize

Connection Interfaces

Read data using the following functions when applies:
- *file*: opens a connection to a file
- *gzfile*: opens a connection to a file compressed with gzip
- *bzfile*: opens a connection to a file compressed with bzip2
- *url*: opens a connection to a webpage

Extract Subsets of Objects

Operator []

Always return an object of the same class as the original, and can be used to select more than one element.

Operator [[]]

Used to extract elements from a list or a data frame. It can only be used to extract a single element, and the class of the returned object does not have to be a list or a data frame.

Operator $

Used to extract elements of a list or a data frame by name.

Subsetting Vectors: Extract the 3rd element of the character vector j.

```
j<-c("a", "b", "c", "d")
j[3]
```

```
## [1] "c"
```

```
j[1:2]
```

```
## [1] "a" "b"
```

Another Example: Using logical index for subset.

```
j[j>"a"]
```

```
## [1] "b" "c" "d"
```

One more example: Creating a logical vector.

```
r<- j>"a"
r
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

```
j[r]
```

```
## [1] "b" "c" "d"
```

Subsetting Lists:

```
c<-list(foo=1:4, bar=0.6)
c
```

```
## $foo
## [1] 1 2 3 4
##
## $bar
## [1] 0.6
```

```
c[1]
```

```
## $foo
## [1] 1 2 3 4
```

```
c[[1]]
```

```
## [1] 1 2 3 4
```

With single brackets, we get a list of sequence 1 to 4; With double brackets, we get only the sequence 1 to 4.

Subsetting Nested Elements of a List

```
o<-list(a=list(10,12,14), b=c(3.14, 2.81))
o
```

```
## $a
## $a[[1]]
## [1] 10
##
## $a[[2]]
## [1] 12
##
## $a[[3]]
## [1] 14
##
##
## $b
## [1] 3.14 2.81
```

```
o[[c(1,3)]]
```

```
## [1] 14
```

```
o[[c(2,2)]]
```

```
## [1] 2.81
```

Subsetting a Matrix

```
xx<-matrix(1:6, 2, 3)
xx
```

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

```
# Select 2nd row, 1st column
xx[2,1]
```

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

```
# Select the 1st row
xx [1,]
```

```
## [1] 1 3 5
```

```
# Select the 3rd column
xx [,3]
```

```
## [1] 5 6
```

Remove NA Values

```
yy<-c(1,2,NA,4,NA,5)
bad<-is.na(yy)
yy[!bad]
```

```
## [1] 1 2 4 5
```

Use complete.cases() function to remove missing values in multiples vectors with missing values.

```
zz<-c("a","b",NA,"d",NA,"f")
good<- complete.cases(yy,zz)
good
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
```

```
yy[good]
```

```
## [1] 1 2 4 5
```

```
zz[good]
```

```
## [1] "a" "b" "d" "f"
```

Vectorized Matrix Operations

```
aa<- matrix(1:4, 2,2); bb<-matrix(rep(10,4), 2, 2)
aa
```

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

```
bb
```

```
##      [,1] [,2]
## [1,]   10   10
## [2,]   10   10
```

```
# Matrix multiplication and division
aa+bb
```

```
##      [,1] [,2]
## [1,]   11   13
## [2,]   12   14
```

```
aa %*%bb #real matrix multiplication
```

```
##      [,1] [,2]
## [1,]   40   40
## [2,]   60   60
```