R-Tutorial 1

Liyuan Dong (dongliyuan@cuhk.edu.cn)

Daojing Zhai (zhaidaojing@cuhk.edu.cn)

09/09/2019

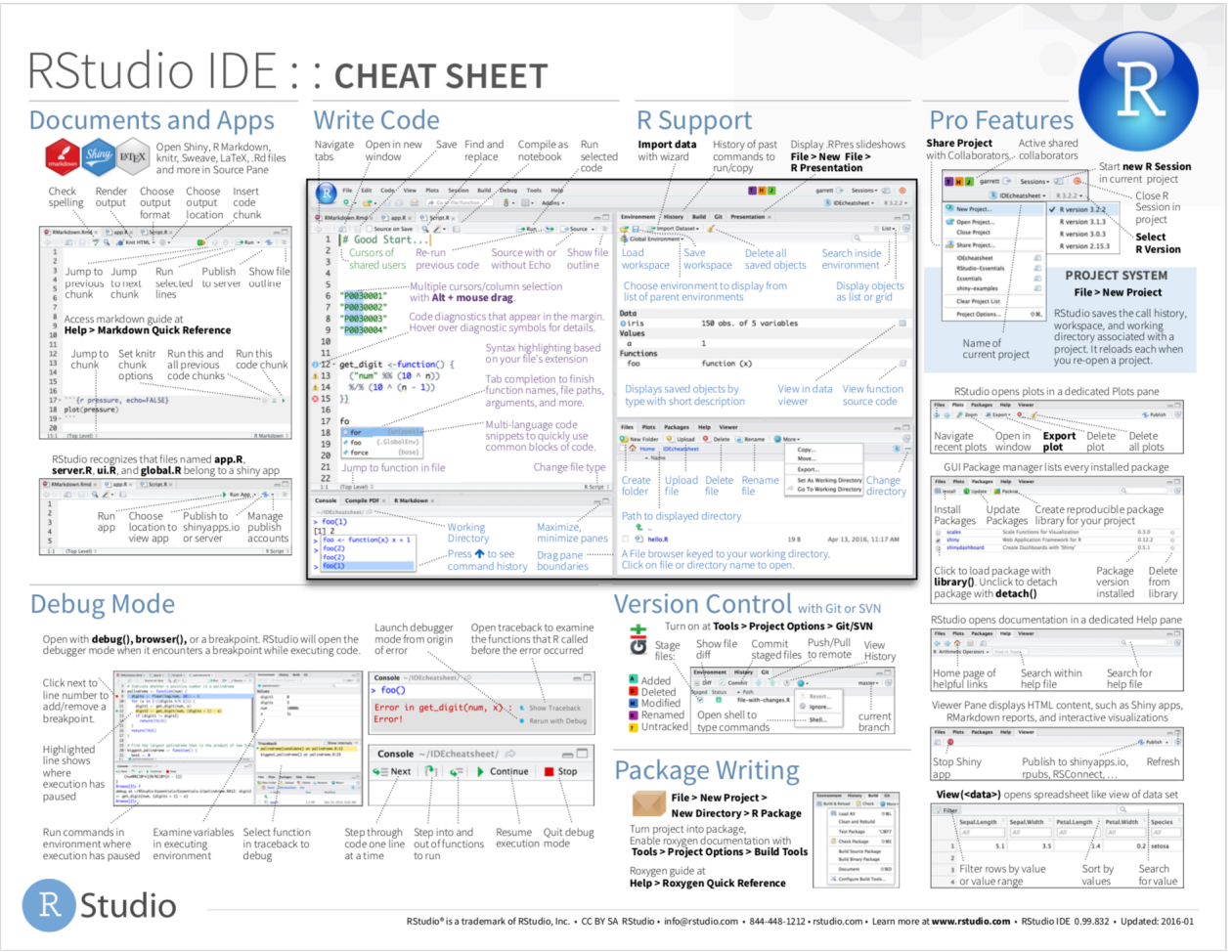
Outline

- Brief introduction to RStudio
- Brief introduction to R language

1.1 Types of Files in R

- Files with the extension *.R*
  - *Files contain the sequence of commands. (script)*
- Files with extension *.RData*
  - *Files that store data in R native format.*
- Files with extension *.Rmd, .md,* and *.Rnw*
  - *R markdown formats.*
- Files with extension *.Rproj*
  - *Editing projects in RStudio.*

1.2 R cheat sheet



RStudio IDE : : CHEAT SHEET

Documents and Apps

Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane

Check spelling | Render output | Choose output format | Choose output location | Insert code chunk

Jump to previous chunk | Jump to next chunk | Run selected lines | Publish to server | Show file outline

Access markdown guide at **Help > Markdown Quick Reference**

Jump to chunk | Set knitr chunk options | Run this and all previous code chunks | Run this code chunk

RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

Run app | Choose location to view app | Publish to shinyapps.io or server | Manage publish accounts

Write Code

Navigate tabs | Open in new window | Save | Find and replace | Compile as notebook | Run selected code

Cursors of shared users | Re-run previous code | Source with or without Echo | Show file outline

Multiple cursors/column selection with **Alt + mouse drag**.

Code diagnostics that appear in the margin. Hover over diagnostic symbols for details.

Syntax highlighting based on your file's extension

Tab completion to finish function names, file paths, arguments, and more.

Multi-language code snippets to quickly use common blocks of code.

Jump to function in file | Change file type

Working Directory | Maximize, minimize panes

Press ↑ to see command history | Drag pane boundaries

R Support

Import data with wizard | History of past commands to run/copy | Display .RPres slideshows **File > New File > R Presentation**

Load workspace | Save workspace | Delete all saved objects | Search inside environment

Choose environment to display from list of parent environments | Display objects as list or grid

Displays saved objects by type with short description | View in data viewer | View function source code

Create folder | Upload file | Delete file | Rename file | Change directory

Path to displayed directory

A File browser keyed to your working directory. Click on file or directory name to open.

Pro Features

Share Project with Collaborators | Active shared collaborators

Start **new R Session** in current project | Close R Session in project | **Select R Version**

PROJECT SYSTEM

File > New Project

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.

Name of current project

RStudio opens plots in a dedicated Plots pane

Navigate recent plots | Open in window | **Export plot** | Delete plot | Delete all plots

GUI Package manager lists every installed package

Install Packages | Update Packages | Create reproducible package library for your project

Click to load package with **library()**. Unclick to detach package with **detach()**

Package version installed | Delete from library

RStudio opens documentation in a dedicated Help pane

Home page of helpful links | Search within help file | Search for help file

Viewer Pane displays HTML content, such as Shiny apps, RMarkdown reports, and interactive visualizations

Stop Shiny app | Publish to shinyapps.io, rpubs, RSConnect, ... | Refresh

View(<data>) opens spreadsheet like view of data set

Filter rows by value or value range | Sort by values | Search for value

Debug Mode

Open with **debug()**, **browser()**, or a breakpoint. RStudio will open the debugger mode when it encounters a breakpoint while executing code.

Launch debugger mode from origin of error | Open traceback to examine the functions that R called before the error occurred

Click next to line number to add/remove a breakpoint.

Highlighted line shows where execution has paused

Run commands in environment where execution has paused | Examine variables in executing environment | Select function in traceback to debug | Step through code one line at a time | Step into and out of functions to run | Resume execution | Quit debug mode

Version Control with Git or SVN

Turn on at **Tools > Project Options > Git/SVN**

Stage files | Show file diff | Commit staged files | Push/Pull to remote | View History

Added | Deleted | Modified | Renamed | Untracked

Open shell to type commands | current branch

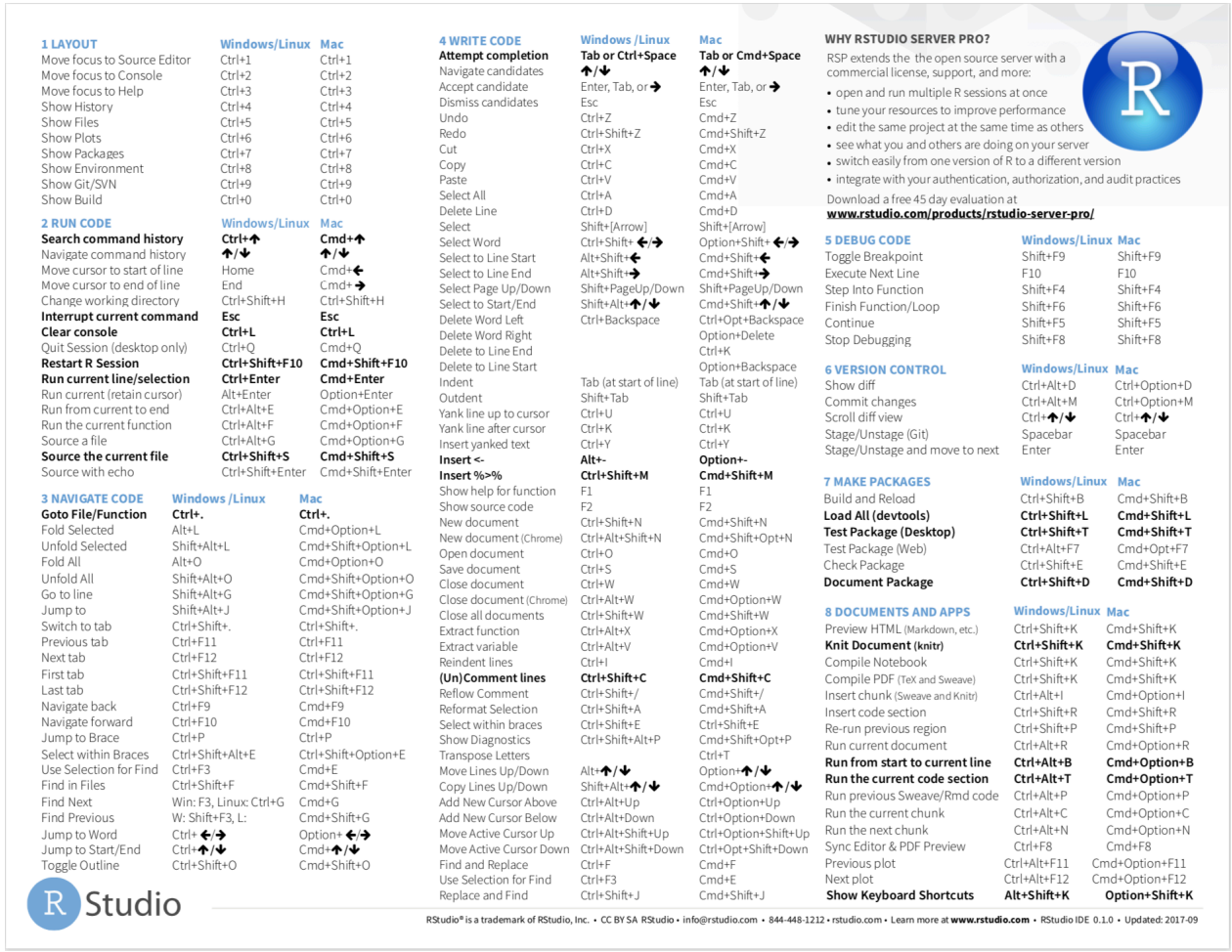
Package Writing

File > New Project > New Directory > R Package

Turn project into package. Enable roxygen documentation with **Tools > Project Options > Build Tools**

Roxygen guide at **Help > Roxygen Quick Reference**

RStudio



1 LAYOUT

	Windows/Linux	Mac
Move focus to Source Editor	Ctrl+1	Ctrl+1
Move focus to Console	Ctrl+2	Ctrl+2
Move focus to Help	Ctrl+3	Ctrl+3
Show History	Ctrl+4	Ctrl+4
Show Files	Ctrl+5	Ctrl+5
Show Plots	Ctrl+6	Ctrl+6
Show Packages	Ctrl+7	Ctrl+7
Show Environment	Ctrl+8	Ctrl+8
Show Git/SVN	Ctrl+9	Ctrl+9
Show Build	Ctrl+0	Ctrl+0

2 RUN CODE

	Windows/Linux	Mac
Search command history	**Ctrl+↑**	**Cmd+↑**
Navigate command history	↑/↓	↑/↓
Move cursor to start of line	Home	Cmd+←
Move cursor to end of line	End	Cmd+→
Change working directory	Ctrl+Shift+H	Ctrl+Shift+H
Interrupt current command	**Esc**	**Esc**
Clear console	**Ctrl+L**	**Ctrl+L**
Quit Session (desktop only)	Ctrl+Q	Cmd+Q
Restart R Session	**Ctrl+Shift+F10**	**Cmd+Shift+F10**
Run current line/selection	**Ctrl+Enter**	**Cmd+Enter**
Run current (retain cursor)	Alt+Enter	Option+Enter
Run from current to end	Ctrl+Alt+E	Cmd+Option+E
Run the current function	Ctrl+Alt+F	Cmd+Option+F
Source a file	Ctrl+Alt+G	Cmd+Option+G
Source the current file	**Ctrl+Shift+S**	**Cmd+Shift+S**
Source with echo	Ctrl+Shift+Enter	Cmd+Shift+Enter

3 NAVIGATE CODE

	Windows /Linux	Mac
Goto File/Function	**Ctrl+.**	**Ctrl+.**
Fold Selected	Alt+L	Cmd+Option+L
Unfold Selected	Shift+Alt+L	Cmd+Shift+Option+L
Fold All	Alt+O	Cmd+Option+O
Unfold All	Shift+Alt+O	Cmd+Shift+Option+O
Go to line	Shift+Alt+G	Cmd+Shift+Option+G
Jump to	Shift+Alt+J	Cmd+Shift+Option+J
Switch to tab	Ctrl+Shift+.	Ctrl+Shift+.
Previous tab	Ctrl+F11	Ctrl+F11
Next tab	Ctrl+F12	Ctrl+F12
First tab	Ctrl+Shift+F11	Ctrl+Shift+F11
Last tab	Ctrl+Shift+F12	Ctrl+Shift+F12
Navigate back	Ctrl+F9	Cmd+F9
Navigate forward	Ctrl+F10	Cmd+F10
Jump to Brace	Ctrl+P	Ctrl+P
Select within Braces	Ctrl+Shift+Alt+E	Ctrl+Shift+Option+E
Use Selection for Find	Ctrl+F3	Cmd+E
Find in Files	Ctrl+Shift+F	Cmd+Shift+F
Find Next	Win: F3, Linux: Ctrl+G	Cmd+G
Find Previous	W: Shift+F3, L:	Cmd+Shift+G
Jump to Word	Ctrl+←/→	Option+←/→
Jump to Start/End	Ctrl+↑/↓	Cmd+↑/↓
Toggle Outline	Ctrl+Shift+O	Cmd+Shift+O

4 WRITE CODE

	Windows /Linux	Mac
Attempt completion	**Tab or Ctrl+Space**	**Tab or Cmd+Space**
Navigate candidates	↑/↓	↑/↓
Accept candidate	Enter, Tab, or→	Enter, Tab, or→
Dismiss candidates	Esc	Esc
Undo	Ctrl+Z	Cmd+Z
Redo	Ctrl+Shift+Z	Cmd+Shift+Z
Cut	Ctrl+X	Cmd+X
Copy	Ctrl+C	Cmd+C
Paste	Ctrl+V	Cmd+V
Select All	Ctrl+A	Cmd+A
Delete Line	Ctrl+D	Cmd+D
Select	Shift+[Arrow]	Shift+[Arrow]
Select Word	Ctrl+Shift+←/→	Option+Shift+←/→
Select to Line Start	Alt+Shift+←	Cmd+Shift+←
Select to Line End	Alt+Shift+→	Cmd+Shift+→
Select Page Up/Down	Shift+PageUp/Down	Shift+PageUp/Down
Select to Start/End	Shift+Alt+↑/↓	Cmd+Shift+↑/↓
Delete Word Left	Ctrl+Backspace	Ctrl+Opt+Backspace
Delete Word Right		Option+Delete
Delete to Line End		Ctrl+K
Delete to Line Start		Option+Backspace
Indent	Tab (at start of line)	Tab (at start of line)
Outdent	Shift+Tab	Shift+Tab
Yank line up to cursor	Ctrl+U	Ctrl+U
Yank line after cursor	Ctrl+K	Ctrl+K
Insert yanked text	Ctrl+Y	Ctrl+Y
Insert <-	**Alt+-**	**Option+-**
Insert %>%	**Ctrl+Shift+M**	**Cmd+Shift+M**
Show help for function	F1	F1
Show source code	F2	F2
New document	Ctrl+Shift+N	Cmd+Shift+N
New document (Chrome)	Ctrl+Alt+Shift+N	Cmd+Shift+Opt+N
Open document	Ctrl+O	Cmd+O
Save document	Ctrl+S	Cmd+S
Close document	Ctrl+W	Cmd+W
Close document (Chrome)	Ctrl+Alt+W	Cmd+Option+W
Close all documents	Ctrl+Shift+W	Cmd+Shift+W
Extract function	Ctrl+Alt+X	Cmd+Option+X
Extract variable	Ctrl+Alt+V	Cmd+Option+V
Reindent lines	Ctrl+I	Cmd+I
(Un)Comment lines	**Ctrl+Shift+C**	**Cmd+Shift+C**
Reflow Comment	Ctrl+Shift+/	Cmd+Shift+/
Reformat Selection	Ctrl+Shift+A	Cmd+Shift+A
Select within braces	Ctrl+Shift+E	Ctrl+Shift+E
Show Diagnostics	Ctrl+Shift+Alt+P	Cmd+Shift+Opt+P
Transpose Letters		Ctrl+T
Move Lines Up/Down	Alt+↑/↓	Option+↑/↓
Copy Lines Up/Down	Shift+Alt+↑/↓	Cmd+Option+↑/↓
Add New Cursor Above	Ctrl+Alt+Up	Ctrl+Option+Up
Add New Cursor Below	Ctrl+Alt+Down	Ctrl+Option+Down
Move Active Cursor Up	Ctrl+Alt+Shift+Up	Ctrl+Option+Shift+Up
Move Active Cursor Down	Ctrl+Alt+Shift+Down	Ctrl+Opt+Shift+Down
Find and Replace	Ctrl+F	Cmd+F
Use Selection for Find	Ctrl+F3	Cmd+E
Replace and Find	Ctrl+Shift+J	Cmd+Shift+J

WHY RSTUDIO SERVER PRO?

RSP extends the the open source server with a commercial license, support, and more:

- open and run multiple R sessions at once
- tune your resources to improve performance
- edit the same project at the same time as others
- see what you and others are doing on your server
- switch easily from one version of R to a different version
- integrate with your authentication, authorization, and audit practices

Download a free 45 day evaluation at **www.rstudio.com/products/rstudio-server-pro/**

5 DEBUG CODE

	Windows/Linux	Mac
Toggle Breakpoint	Shift+F9	Shift+F9
Execute Next Line	F10	F10
Step Into Function	Shift+F4	Shift+F4
Finish Function/Loop	Shift+F6	Shift+F6
Continue	Shift+F5	Shift+F5
Stop Debugging	Shift+F8	Shift+F8

6 VERSION CONTROL

	Windows/Linux	Mac
Show diff	Ctrl+Alt+D	Ctrl+Option+D
Commit changes	Ctrl+Alt+M	Ctrl+Option+M
Scroll diff view	Ctrl+↑/↓	Ctrl+↑/↓
Stage/Unstage (Git)	Spacebar	Spacebar
Stage/Unstage and move to next	Enter	Enter

7 MAKE PACKAGES

	Windows/Linux	Mac
Build and Reload	Ctrl+Shift+B	Cmd+Shift+B
Load All (devtools)	**Ctrl+Shift+L**	**Cmd+Shift+L**
Test Package (Desktop)	**Ctrl+Shift+T**	**Cmd+Shift+T**
Test Package (Web)	Ctrl+Alt+F7	Cmd+Opt+F7
Check Package	Ctrl+Shift+E	Cmd+Shift+E
Document Package	**Ctrl+Shift+D**	**Cmd+Shift+D**

8 DOCUMENTS AND APPS

	Windows/Linux	Mac
Preview HTML (Markdown, etc.)	Ctrl+Shift+K	Cmd+Shift+K
Knit Document (knitr)	**Ctrl+Shift+K**	**Cmd+Shift+K**
Compile Notebook	Ctrl+Shift+K	Cmd+Shift+K
Compile PDF (TeX and Sweave)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave and Knitr)	Ctrl+Alt+I	Cmd+Option+I
Insert code section	Ctrl+Shift+R	Cmd+Shift+R
Re-run previous region	Ctrl+Shift+P	Cmd+Shift+P
Run current document	Ctrl+Alt+R	Cmd+Option+R
Run from start to current line	**Ctrl+Alt+B**	**Cmd+Option+B**
Run the current code section	**Ctrl+Alt+T**	**Cmd+Option+T**
Run previous Sweave/Rmd code	Ctrl+Alt+P	Cmd+Option+P
Run the current chunk	Ctrl+Alt+C	Cmd+Option+C
Run the next chunk	Ctrl+Alt+N	Cmd+Option+N
Sync Editor & PDF Preview	Ctrl+F8	Cmd+F8
Previous plot	Ctrl+Alt+F11	Cmd+Option+F11
Next plot	Ctrl+Alt+F12	Cmd+Option+F12
Show Keyboard Shortcuts	**Alt+Shift+K**	**Option+Shift+K**

RStudio

I will send you later.

Suggestion: armed with a cookbook and a cheat sheet, practice and practice!

2. Brief introduction to R language

- Data type
- Basic Operator
- Data structure
- Basic Logic
- Basic functions

2.1 Basic Data type

- Character:
  - ‘a’, ‘swc’
- Numeric (real or decimal)
  - 1.4, 1
- Logical (TRUE/FALSE)

```
> # character
> x <- 'a'
> y <- 'swc'
> typeof(x)
[1] "character"
> print(x)
[1] "a"
> # numeric
> x <- 1.4
> y <- 1
> print(x)
[1] 1.4
> print(y)
[1] 1
> typeof(x)
[1] "double"
> # Logical
> x <- TRUE
> typeof(x)
[1] "logical"
> y <- FALSE
> typeof(y)
[1] "logical"
```

Use ‘typeof()’ or ‘class()’ to see the data type(class)

2.2 Basic Operator

- Arithmetic operators (you must be familiar with it:)
  - $+, -, *, /, \%/\%$

```
> x <- 5
> y <- 16
> x+y
[1] 21
> x-y
[1] -11
> x*y
[1] 80
> y/x
[1] 3.2
> # Modulus
>
> y%/%x
[1] 3
> y^x
[1] 1048576
```

2.2 Basic Operator

- Arithmetic operators
  - $+, -, *, /, \%/\%$
- Rational operators
  - $<, >, \leq, \geq, ==, !=$

```
>  x <- 5
>  y <- 16
> x<y
[1] TRUE
> x>y
[1] FALSE
> x<=5
[1] TRUE
> y>=20
[1] FALSE
> y==20
[1] FALSE
> y==16
[1] TRUE
> x!=5
[1] FALSE
```

2.2 Basic Operator

- Arithmetic operators
  - $+, -, *, /, \%/\%$
- Rational operators
  - $<, >, \leq, \geq, ==, !=$
- Logical operators
  - !, &, &&, |, ||
- Assignment
  - <-, =
  - Take care of '==' and '='!

```
> x <- c(TRUE,FALSE,0,6)
> y <- c(FALSE,TRUE,FALSE,TRUE)
> # Logical NOT
> !x
[1] FALSE  TRUE  TRUE FALSE
> # Element-wise logical AND
> x&y
[1] FALSE FALSE FALSE  TRUE
> # Logical AND
> x&&y
[1] FALSE
> # Element-wise logical OR
> x|y
[1]  TRUE  TRUE FALSE  TRUE
> # Logical OR
> x||y
[1] TRUE
```

2.3 Basic Data Structure

- Vector
- Data Frame
- List
- Matrices

2.3 Basic Data Structure

- Vector
  - Combination function “c()”

```
# create numeric atomic vector
x <- c(1,2,3)

# print it
print(x)

## [1] 1 2 3

# create character atomic vector
y <- c('text 1', 'text 2', 'text 3', 'text 4')

# print it
print(y)

## [1] "text 1" "text 2" "text 3" "text 4"

# a mixed vector
x <- c(1, 2, '3')

# print result of forced conversion
print(x)

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

2.3 Basic Data Structure

- Vector
  - Combination function “c()”
  - Create vector element by element

```
# create numeric vectors
x <- 1:5
y <- 2:6

# print sum
print(x+y)

## [1]  3  5  7  9 11

# print multiplication
print(x*y)

## [1]  2  6 12 20 30

# print division
print(x/y)

## [1] 0.5000000 0.6666667 0.7500000 0.8000000 0.8333333

# print exponentiation
print(x^y)

## [1]     1     8    81  1024 15625
```

2.3 Basic Data Structure

- Vector
  - Combination function “c()”
  - Create vector element by element
  - Create vector with names

```
# create named vector
x <- c(item1 = 10, item2 = 14, item3 = 9, item4 = 2)

# print it
print(x)

## item1 item2 item3 item4
##    10    14     9     2

# create unnamed vector
x <- c(10, 14, 9, 2)

# set names of elements
names(x) <- c('item1', 'item2', 'item3', 'item4')

# print it
print(x)

## item1 item2 item3 item4
##    10    14     9     2
```

2.3 Basic Data Structure

- Vector
  - Combination function “c()”
  - Create vector element by element
  - Create vector with names
  - Create sequence with seq

```
# create sequence with seq
my.seq <- seq(from = -10, to = 10, by = 2)

# print it
print(my.seq)
```

```
##  [1] -10  -8  -6  -4  -2   0   2   4   6   8  10
```

```
# create sequence with defined number of elements
my.seq <- seq(from = 0, to = 10, length.out = 20)

# print it
print(my.seq)
```

```
##  [1]  0.0000000  0.5263158  1.0526316  1.5789474  2.1052632
##  [6]  2.6315789  3.1578947  3.6842105  4.2105263  4.7368421
## [11]  5.2631579  5.7894737  6.3157895  6.8421053  7.3684211
## [16]  7.8947368  8.4210526  8.9473684  9.4736842 10.0000000
```

2.3 Basic Data Structure

- Vector
  - Combination function “c()”
  - Create vector element by element
  - Create vector with names
  - Create sequence with seq
  - Generate data from distribution

```
# generate 10 random numbers from a Normal distribution
my.rnd.vec <- rnorm(n = 10, mean = 0, sd = 1)

# print it
print(my.rnd.vec)
```

```
##  [1]  0.60458285  0.24434564  0.12614266 -1.47225224
##  [5]  0.17259734  0.88561239  0.28164064 -0.23263138
##  [9] -0.06545415  1.61985720
```

```
# create a random vector with minimum and maximum
my.rnd.vec <- runif(n = 10, min = -5, max = 5)

# print it
print(my.rnd.vec)
```

```
##  [1] -4.2295006 -4.1491707  2.5542791  1.6543161 -0.9159421
##  [6]  2.6246907 -3.9998592 -1.3303270  3.2227269  1.1148276
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe

	ticker	date	prices
1	AAP	2010-01-04	40.38
2	AAP	2010-01-05	40.14
3	AAP	2010-01-06	40.49
4	AAP	2010-01-07	40.48
5	AAP	2010-01-08	40.64
6	COG	2010-01-04	46.23
7	COG	2010-01-05	46.17
8	COG	2010-01-06	45.97
9	COG	2010-01-07	45.56
10	COG	2010-01-08	45.46
11	BLK	2010-01-04	238.58
12	BLK	2010-01-05	239.61
13	BLK	2010-01-06	234.67
14	BLK	2010-01-07	237.25
15	BLK	2010-01-08	238.92
16	CAM	2010-01-04	43.43
17	CAM	2010-01-05	43.96
18	CAM	2010-01-06	44.26
19	CAM	2010-01-07	44.50
20	CAM	2010-01-08	44.86

A Data Frame Example: my.df
Please Load the file in your workplace

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
    - Column Name
    - Get column
    - Access specific column/row

```
# get names of columns with names
names(my.df)

## [1] "ticker" "date"   "prices"

# get names of columns with colnames
colnames(my.df)

## [1] "ticker" "date"   "prices"

# get column ticker from my.df
my.ticker <- my.df$ticker

# get column price from my.df
my.prices <- my.df['prices']

# get second column from my.df
my.date <- my.df[ ,2]
# accessing rows 1:5, all columns
print(my.df[1:5, ])
# selecting rows 1 to 3, columns 'ticker' and 'prices'
print(my.df[1:3, c('ticker','prices')])
# accessing rows 1:5, columns 1 and 2
print(my.df[1:5, c(1,2)])
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
  - Modify a Dataframe
    - Add new columns
    - Rename column names
    - Remove columns
    - Find index

```r
# add a sequence to my.df
my.df$my.seq <- 1:nrow(my.df)

# set new col by name
my.df['my.seq.2'] <- seq(1,100, length.out = nrow(my.df))

# set new col by position
my.df[[6]] <- seq(1,10, length.out = nrow(my.df))

# rename colnames
colnames(my.df) <- c('ticker', 'date', 'prices',
                     'my.seq', 'my.seq.2', 'my.seq.3')

# removing columns
my.df$my.seq <- NULL
my.df$my.seq.2 <- NULL
my.df$my.seq.3 <- NULL
my.df$V6 <- NULL
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
  - Modify a Dataframe
    - Add new columns
    - Rename column names
    - Remove columns
    - Find index

```r
# add a sequence to my.df
my.df$my.seq <- 1:nrow(my.df)

# set new col by name
my.df['my.seq.2'] <- seq(1,100, length.out = nrow(my.df))

# set new col by position
my.df[[6]] <- seq(1,10, length.out = nrow(my.df))

# rename colnames
colnames(my.df) <- c('ticker', 'date', 'prices',
                     'my.seq', 'my.seq.2', 'my.seq.3')

# removing columns
my.df$my.seq <- NULL
my.df$my.seq.2 <- NULL
my.df$my.seq.3 <- NULL
my.df$V6 <- NULL
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
  - Modify a Dataframe
    - Add new columns
    - Rename column names
    - Remove columns
    - Find index

```r
# add a sequence to my.df
my.df$my.seq <- 1:nrow(my.df)

# set new col by name
my.df['my.seq.2'] <- seq(1,100, length.out = nrow(my.df))

# set new col by position
my.df[[6]] <- seq(1,10, length.out = nrow(my.df))

# rename colnames
colnames(my.df) <- c('ticker', 'date', 'prices',
                     'my.seq', 'my.seq.2', 'my.seq.3')

# removing columns
my.df$my.seq <- NULL
my.df$my.seq.2 <- NULL
my.df$my.seq.3 <- NULL
my.df$V6 <- NULL
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
  - Modify a Dataframe
  - Sort a Dataframe
    - Sort Dataframe
    - Order Dataframe by one col
    - Order Dataframe by several cols

```r
# set index with positions of ascending order in col1
idx <- order(my.df$col1)

# print it
print(idx)

# order my.df by col1
my.df.2 <- my.df[order(my.df$col1), ]

# print result
print(my.df.2)

# sort df with col2 and col1
my.df.3 <- my.df[order(my.df$col2, my.df$col1), ]

# print result
print(my.df.3)
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
  - Modify a Dataframe
  - Sort a Dataframe
  - Combine and aggregate Dataframe
    - Join dataframes by rows/columns

Q: What if two dataframes have different size?

```r
# set two dfs with same colnames
my.df.1 <- data.frame(col1 = 1:5, col2 = rep('a', 5))
my.df.2 <- data.frame(col1 = 6:10, col2 = rep('b', 5))

# bind them by rows
my.df <- rbind(my.df.1, my.df.2)

# print result
print(my.df)
```

```
##    col1 col2
## 1     1    a
## 2     2    a
## 3     3    a
## 4     4    a
## 5     5    a
## 6     6    b
## 7     7    b
## 8     8    b
## 9     9    b
## 10   10    b
```

```r
# column bind dfs
my.df <- cbind(my.df.1, my.df.2)

# print result
print(my.df)
```

```
##   col1 col2 col3 col4
## 1    1    a    6    b
## 2    2    a    7    b
## 3    3    a    8    b
## 4    4    a    9    b
## 5    5    a   10    b
```

2.3 Basic Data Structure

- Vector
- Data Frame
  - Create a Dataframe
  - Access Information
  - Modify a Dataframe
  - Sort a Dataframe
  - Combine and aggregate Dataframe
    - Join dataframes by rows/columns
    - Merge two dataframes

Q: What if two dataframes have different size?

```r
# set dfs
my.df.1 <- data.frame(date = as.Date('2016-01-01')+0:10,
                      x = 1:11)

my.df.2 <- data.frame(date = as.Date('2016-01-05')+0:10,
                      y = seq(20,30, length.out = 11))

# merge dfs by date
my.df <- merge(my.df.1, my.df.2, by = 'date')

# print result
print(my.df)
```

```
##         date  x  y
## 1 2016-01-05  5 20
## 2 2016-01-06  6 21
## 3 2016-01-07  7 22
## 4 2016-01-08  8 23
## 5 2016-01-09  9 24
## 6 2016-01-10 10 25
## 7 2016-01-11 11 26
```

2.3 Basic Data Structure

- Vector
- Data Frame
- List
  - Create a list (fun: *list*)
    - Without/with item names
  - Access the elements
    - Method 1: index (‘[[ ]]’ vs. ‘[ ]’)

```
# set list
my.l <- list(2, 1:5, c('a', 'b'))

# create named list
my.named.l <- list(ticker = 'ABC',
                   name.company = 'Company ABC',
                   price = c(1,1.5,2,2.3),
                   market = 'NYSE',
                   date.price = as.Date('2016-01-01')+0:3)

# print second element of my.l
print(my.l[[2]])

## [1] 1 2 3 4 5

# accessing list with [[ ]]
class(my.l[[2]])

## [1] "integer"

# accessing list with [ ]
class(my.l[2])

## [1] "list"
```

2.3 Basic Data Structure

- Vector
- Data Frame
- List
  - Create a list (fun: *list*)
    - Without/with item names
  - Access the elements
    - Method 1: index ('[[ ]]' vs. '[ ]')
    - Method 2: '$'
    - Method 3: name '[[ ]]'

```
# set list
my.l <- list(2, 1:5, c('a', 'b'))

# create named list
my.named.l <- list(ticker = 'ABC',
                   name.company = 'Company ABC',
                   price = c(1,1.5,2,2.3),
                   market = 'NYSE',
                   date.price = as.Date('2016-01-01')+0:3)

# print second element of my.l
print(my.l[[2]])

## [1] 1 2 3 4 5

# accessing elements of a list using $
print(my.named.l$ticker)

## [1] "ABC"

# accessing elements of a list using [['name']]
print(my.named.l[['ticker']])

## [1] "ABC"
```

2.3 Basic Data Structure

- Vector
- Data Frame
- List
  - Create a list (fun: *list*)
  - Access the elements
  - Add/remove elements

```
# set list
my.l <- list('a',1,3)

# change value at position 4
my.l[[4]] <- c(1:5)

# change value at position 2
my.l[[2]] <- c('b')

# remove third element
my.l[[3]] <- NULL

# set list
my.l <- list(1, 2, 3, 4)

# remove all elements higher than 2
my.l[my.l > 2] <- NULL

# set list
my.l <- list(1, 2, 3, 4, 'a', 'b')

# print logical test (return NA value)
print(my.l > 2)

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

2.3 Basic Data Structure

- Vector
- Data Frame
- List
- Matrices
  - Create a matrix

```
# set raw data with prices
raw.data <- c(40.38,  40.14,  40.49,  40.48,  40.64,
              46.23,  46.17,  45.97,  45.56,  45.46,
              238.58, 239.61, 234.67, 237.25, 238.92,
              43.43,  43.96,  44.26,  44.5,   44.86)

# create matrix
my.mat <- matrix(raw.data, nrow = 5, ncol = 4)
colnames(my.mat) <- c('AAP', 'COG', 'BLK', 'CAM')
rownames(my.mat) <- c("2010-01-04", "2010-01-05", "2010-01-06",
                      "2010-01-07", "2010-01-08")

# print result
print(my.mat)
```

```
##                AAP   COG    BLK   CAM
## 2010-01-04 40.38 46.23 238.58 43.43
## 2010-01-05 40.14 46.17 239.61 43.96
## 2010-01-06 40.49 45.97 234.67 44.26
## 2010-01-07 40.48 45.56 237.25 44.50
## 2010-01-08 40.64 45.46 238.92 44.86
```

```
# print the names of columns
print(colnames(my.mat))
```

```
## [1] "AAP" "COG" "BLK" "CAM"
```

```
# print the names of rows
print(rownames(my.mat))
```

```
## [1] "2010-01-04" "2010-01-05" "2010-01-06" "2010-01-07"
## [5] "2010-01-08"
```

Advice: use dataframe for table, matrix for math operation

2.3 Basic Data Structure

- Vector
- Data Frame
- List
- Matrices
  - Create a matrix
  - Select elements

```
# create matrix
my.mat <- matrix(1:9, nrow = 3)

# display it
print(my.mat)

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

# display element in [1,2]
print(my.mat[1,2])

## [1] 4
# select all rows from column 2
print(my.mat[ , 2])

## [1] 4 5 6

# select all columns from row 1
print(my.mat[1, ])

## [1] 1 4 7
# select some elements and print it
print(my.mat[2:3,1:2])
```

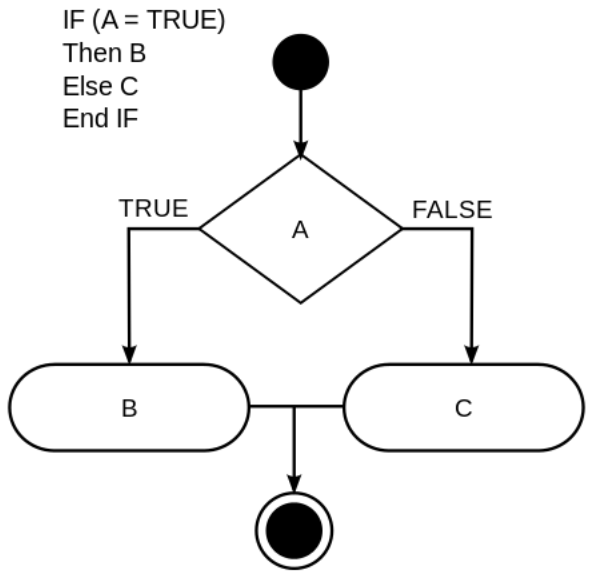
2.3 Basic Data Structure

- Vector
- Data Frame
- List
- Matrices
  - Create a matrix
  - Select elements
  - Useful functions
    - Transpose: *t()*
    - Determinant: *det()*
    - Inverse of a matrix (from *matlib* package): *inv()*
    - Mean of a matrix, row wise: *rowMeans()*
    - Mean of a matrix, column wise: *colMeans()*

```r
# Create a matrix
my.mat <- matrix(c(5,1,0,
                   3,-1,2,
                   4,0,-1),nrow=3,byrow=TRUE)

# Transpose
print(t(my.mat))

# Determinant
print(det(my.mat))

# Inverse (require matlib package)
library(matlib)
print(inv(my.mat))

# row wise mean
print(rowMeans(my.mat))

# column wise mean
print(colMeans(my.mat))
```

2.4. Basic Logic

- Conditional constructs
  - If…else if…else…




```
x <- 0
if (x < 0) {
  print("Negative number")
} else if (x > 0) {
  print("Positive number")
} else
  print("Zero")
```

2.4. Basic Logic

- Conditional constructs
- Loop
  - For loop

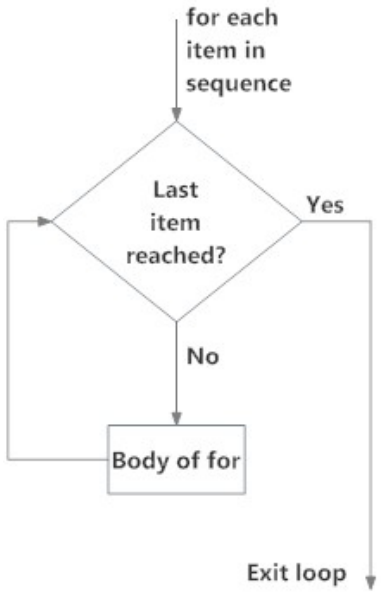


Fig: operation of for loop

```
m = 2
n = 2
for(i in 1:m) {
  for(j in 1:n) {
      print(i*j)
  }
}
```

Q: the output?

A: 1,2,2,4

B: 1,2,4

2.4. Basic Logic

- Conditional constructs
- Loop
  - For loop
    - Break

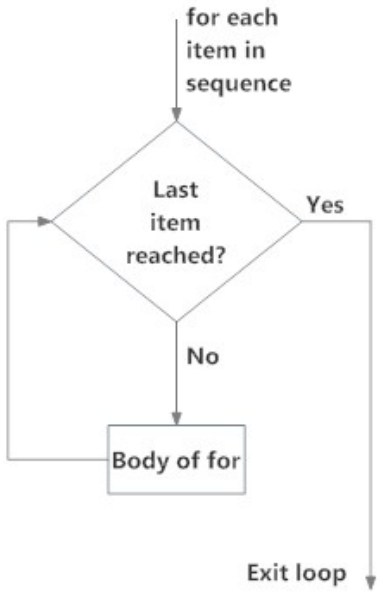


Fig: operation of for loop

```
m = 2
n = 2
for(i in 1:m) {
  for(j in 1:n) {
    if(i==j) {
      break;
    } else {
      print(i*j)
    }
  }
}
```

Q: the output?

A: 2

B: 2,2

C: nothing

2.4. Basic Logic

- Conditional constructs
- Loop
  - For loop
    - Break
    - Next

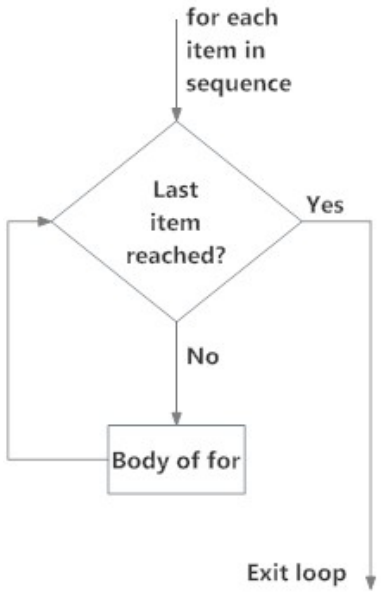


Fig: operation of for loop

```
m = 2
n = 2
for (i in 1:m) {
  for (j in 1:n) {
    if (i==j) {
      next;
    } else {
      print(i*j)
    }
  }
}
```

Q: the output?

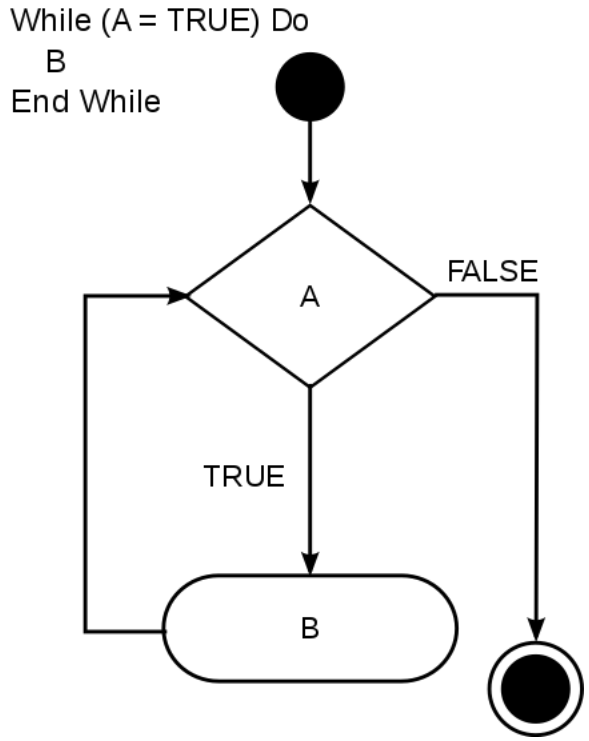
A: 2

B: 2,2

C: nothing

2.4. Basic Logic

- Conditional constructs
- Loop
  - For loop
  - While loop




```
i = 1
j = 1
m = 2
n = 2

while(i<=m){
  while(j<=n){
    print(i*j)
    j=j+1
  }
  i=i+1
}
```

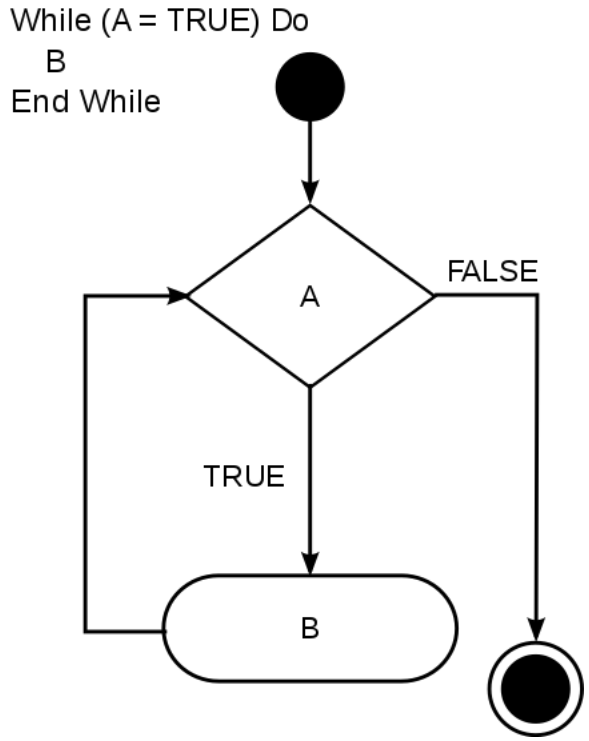
Q: the output?

A: 1,2,2,4

B: 1,2,4

2.4. Basic Logic

- Conditional constructs
- Loop
  - For loop
  - While loop
    - A repeating if statement




```
i = 1
j = 1
m = 2
n = 2

while(i<=m){
  while(j<=n){
    print(i*j)
    j=j+1
  }
  j=1
  i=i+1
}
```

Initialize

Statement

Update

2.4. Basic Logic

- Conditional constructs
- Loop
  - For loop
  - While loop
  - Vectorization (not cover in this course)

2.5. Function

- Create a function
  - Why do we need a function?
    - Applicability: organize the repetitive tasks
    - Easy to fix bug (only change in one place)
  - How to create a function?
    - Input
    - Processing stage
    - Output
  - Call a function in main script

```
my.fct <- function(arg1 = 1, arg2 = 'abc', ...){
  # Description goes here
  #
  # Args:
  #   arg1 - description of arg1 here
  #   arg2 - description of arg2 here
  #
  # Returns:
  #   description of returned object goes here
  ...

  return(out)

}

out <- my.fct(arg1 = 2, arg2 = 'bcd')
```

2.5. Function

- Create a function
  - Why do we need a function?
    - Applicability: organize the repetitive tasks
    - Easy to fix bug (only change in one place)
  - How to create a function?
    - Input
    - Processing stage
    - Output
  - Call a function in main script

```
my.fct <- function(list.input,list.length){
  output = 0
  for (i in 1:list.length){
    output = output + list.input[[i]]
  }
  return(output)
}

sum <- my.fun(1:5,5)
```

2.5. Function

- Create a function
- Build-in function
  - *apply*: alternative to *loops*
    - *lapply:* input a list and return a list
    - *sapply*: return a vector or matrix

```
# set list
my.l <- list(1:10, 2:5, 10:-20)

# use lapply with mean
my.mean.vec <- lapply(X = my.l, FUN = mean)

# print result
print(my.mean.vec)

## [[1]]
## [1] 5.5
##
## [[2]]
## [1] 3.5
##
## [[3]]
## [1] -5

# create list
my.l <- list(1:10, 2:5, 10:-20)

# use sapply
my.mean.vec <- sapply(my.l, mean)

# print result
print(my.mean.vec)

## [1]  5.5  3.5 -5.0
```

2.5. Function

- Create a function
- Build-in function
  - *apply*: alternative to *loops*
    - *lapply:* input a list and return a list
    - *sapply*: return a vector or matrix
    - *apply*: used in 2D objects, by rows/cols

```
# set matrix and print it
my.mat <- matrix(1:15, nrow = 5)

# sum rows with apply and print it
sum.rows <- apply(X = my.mat, MARGIN = 1, FUN = sum)
print(sum.rows)

## [1] 18 21 24 27 30

# sum columns with apply and print it
sum.cols <- apply(X = my.mat, MARGIN = 2, FUN = sum)
print(sum.cols)

## [1] 15 40 65
```

2.5. Function

- Create a function
- Build-in function
  - *apply*: alternative to *loops*
    - *lapply:* input a list and return a list
    - *sapply*: return a vector or matrix
    - *apply*: used in 2D objects, by rows/cols
  - *by:* split a *dataframe* based on a factor

```r
# load data
load('data/SP500-Stocks-WithRet.RData')

# set function for processing df
my.fct <- function(df.in){

  P <- df.in$price.adjusted
  ret <- df.in$ret

  out <- c(MeanPrice= mean(P),
           MaxPrice = max(P),
           MinPrice = min(P),
           MeanRet = mean(ret),
           MaxRet = max(ret),
           MinRet = min(ret))

  return(out)

}

# apply my.fct for each ticker in my.df
my.l <- by(data = my.df, INDICES = my.df$ticker, FUN = my.fct)
```