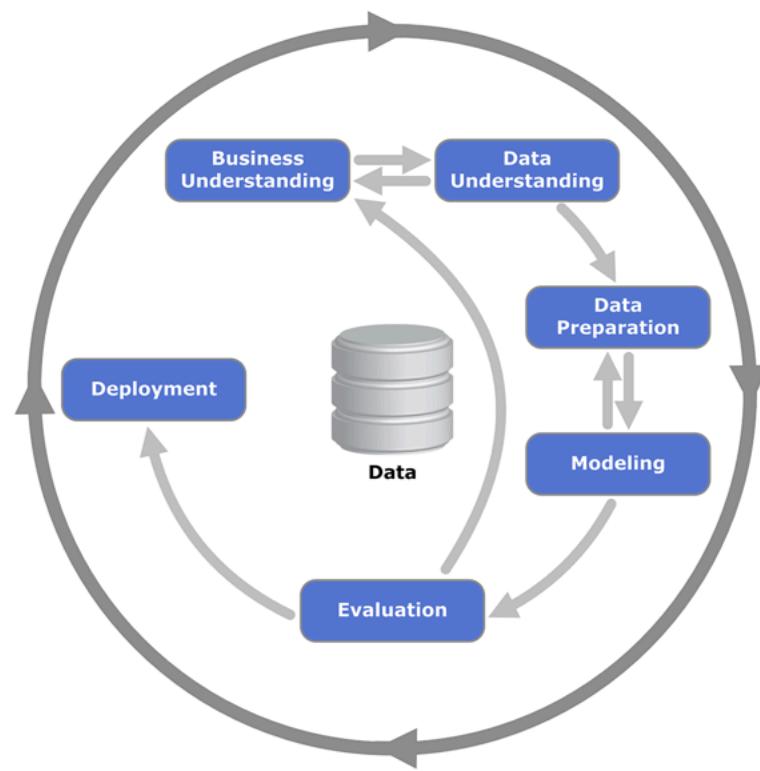


# 1장. 분석

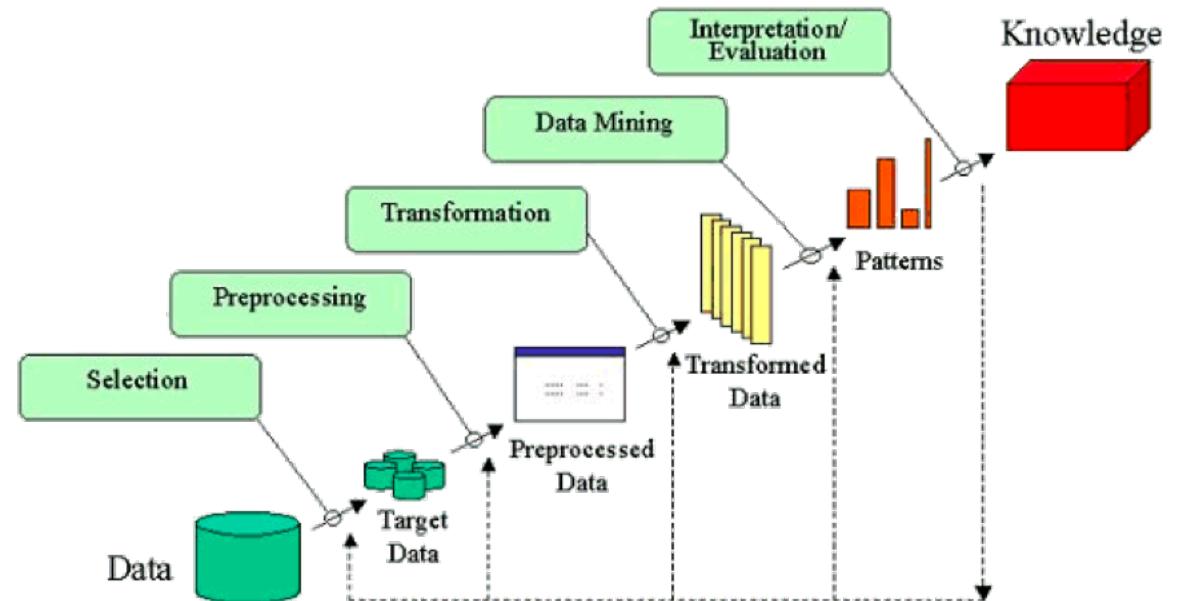
There are only hard things in Computer Science.

Cache Invalidation and naming things

## CRISP-DM



## KDD



<https://cran.r-project.org/bin/windows/base/>

The screenshot shows a web browser window with the URL [cran.r-project.org/bin/windows/base/](https://cran.r-project.org/bin/windows/base/) in the address bar. The page title is "R-3.6.1 for Windows (32/64 bit)". A prominent red dashed box highlights the download link "Download R 3.6.1 for Windows (81 megabytes, 32/64 bit)". Below the link are three blue hyperlinks: "Installation and other instructions", "New features in this version", and "FAQ".

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server. You will need a version of md5sum for windows: both [graphical](#) and [command line versions](#) are available.

#### Frequently asked questions

- [Does R run under my version of Windows?](#)
- [How do I update packages in my previous version of R?](#)
- [Should I run 32-bit or 64-bit R?](#)

Please see the [R FAQ](#) for general information about R and the [R Windows FAQ](#) for Windows-specific information.

#### Other builds

- Patches to this release are incorporated in the [r-patched snapshot build](#).
- A build of the development version (which will eventually become the next major release of R) is available in the [r-devel snapshot build](#).
- [Previous releases](#)

Note to webmasters: A stable link which will redirect to the current Windows binary release is  
<https://<CRAN MIRROR>/bin/windows/base/release.htm>.

<https://www.rstudio.com/products/rstudio/#Desktop>

The screenshot shows the RStudio Desktop product page. At the top, there's a navigation bar with links for Bookmarks, SAS® Logon Mana..., 앱 (App), MO, 번역 (Translation), JSON, syntax, Kolon\_viya, repl.it, 지도좌표변환 (Coordinate Transformation), JVM Thread dump..., 04.데이터 분석 (Data Analysis), 22.데이터시각화 (Data Visualization), LinkForR, and Influencer. Below the navigation is the R Studio logo and a horizontal menu with Products, Resources, Pricing, About Us, Blogs, and a search icon.

The main content area features two tabs: "Open Source Edition" (selected) and "Commercial License".

**Overview** (under Open Source Edition):

- Access RStudio locally
- Syntax highlighting, code completion, and smart indentation
- Execute R code directly from the source editor
- Quickly jump to function definitions
- Easily manage multiple working directories using projects
- Integrated R help and documentation
- Interactive debugger to diagnose and fix errors quickly
- Extensive package development tools

All of the features of open source; plus:

- A commercial license for organizations not able to use AGPL software
- Access to priority support

**Support**: Community forums only

- Priority Email Support
- 8 hour response during business hours (ET)

**License**: AGPL v3

**Pricing**: Free (\$995/year)

**Buttons**: DOWNLOAD RSTUDIO DESKTOP (highlighted with a red dashed box) and BUY NOW.

# Anaconda 설치 – Jupyter Notebook 설치

02. 데이터 탑 및 구조

<https://www.anaconda.com/distribution/>

The screenshot shows the official Anaconda distribution website. At the top, there's a navigation bar with a back button, forward button, refresh button, and a search bar containing 'anaconda.com/distribution/'. Below the search bar is a bookmarks bar with links to SAS Logon Manager, MO, 번역, JSON, syntax, Kolon\_viya, repl.it, JVM Thread dump..., 04.데이터 분석, 22.데이터시각화, LinkForR, Influencer Marketi..., and 기타 북마크.

The main content area features a brief introduction: "data science and machine learning on Linux, Windows, and Mac OS X. With over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:". Below this is a bulleted list of features:

- Quickly download 1,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

To the right of the text, there's a grid of logos for various data science tools: jupyter, spyder, NumPy, SciPy, Numba, pandas, DASK, Bokeh, HoloViews, DataShader, matplotlib, learn, H2O.ai, TensorFlow, and CONDA.

At the bottom, there are download links for Windows, macOS, and Linux:

- Windows | macOS | Linux

The Windows section is highlighted with a red dashed box around the 'Download' button for the Python 3.7 version. It also lists 64-Bit Graphical Installer (486 MB) and 32-Bit Graphical Installer (418 MB).

The Linux section shows a 'Download' button for the Python 2.7 version, listing 64-Bit Graphical Installer (427 MB) and 32-Bit Graphical Installer (361 MB).

```
conda install -c r r-essentials
```

<https://git-scm.com/downloads>

The screenshot shows the main navigation bar with links for About, Documentation, Downloads (selected), and Community. The Downloads section features a large image of a Mac monitor with the text "Latest source Release 2.23.0". Below it are download links for Mac OS X, Windows, and Linux/Unix. A note says "Older releases are available and the Git source repository is on GitHub." On the left, there's a sidebar for the "Pro Git book" and sections for GUI Clients and Logos.

This screenshot shows the "git-scm.com/download/win" page. It displays a progress bar indicating "Your download is starting..." for the latest version (2.23.0). It includes links for other Windows downloads like 32-bit and 64-bit versions, and a link to the "Pro Git book". The "Now What?" section at the bottom encourages users to start using Git.

The screenshot shows a web browser window with the URL <https://translate.google.co.kr/?hl=ko>. The main content is the 'GUI Clients' section of the official Git website. The page has a dark header with the Git logo and navigation links for 'About', 'Documentation', 'Downloads', 'Community', and 'Logos'. A sidebar on the left contains a note about the 'Pro Git book' and links to 'SourceTree', 'GitHub Desktop', 'TortoiseGit', and 'Git Extensions'.

**GUI Clients**

Git comes with built-in GUI tools for committing ([git-gui](#)) and browsing ([gitk](#)), but there are several third-party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just [follow the instructions](#).

**All**    **Windows**    **Mac**    **Linux**    **Android**    **iOS**

**SourceTree**  
Platforms: Mac, Windows  
Price: Free  
License: Proprietary

**GitHub Desktop**  
Platforms: Mac, Windows  
Price: Free  
License: MIT

**TortoiseGit**  
Platforms: Windows  
Price: Free

**Git Extensions**  
Platforms: Linux, Mac, Windows  
Price: Free

## 2장. 데이터 타입 및 데이터 구조

There are only hard things in Computer Science.

Cache Invalidation and naming things

- 변수 이름 규칙

- 알파벳, 숫자, "." 과 "\_" 을 사용 할 수 있다.
- 변수 이름의 첫 문자는 숫자 또는 "\_" 가 될 수 없다.
- 변수의 첫번째 문자가 "." 로 시작하는 경우 대부분 R 의 내부 변수 인 경우가 많다
- R에서는 대/소 문자를 구분한다. (Case Sensitive)
- 올바른 변수 사용 예

```
> x
> x1
> studentCount
> product1
> order_type
> .measure
```

- 올바르지 않은 사용 예

```
> 1x
> .1test
> test-case
```

- <- 또는 = 을 이용해서 변수 할당.
- <- 를 사용 할 것을 권장

## 팁과 조언

<- 은 모든 곳에서 사용 할 수 있는 반면, = 은 사용되는 곳에 제약이 있습니다.

```
> temp1 <- max(x <- c(1,2,3))
> temp1
[1] 3
> x
[1] 1 2 3

> temp2 <- max(y = c(1,2,3))
> temp2
[1] 3
> y
Error: object 'y' not found
```

- 다양한 변수 할당 방법

- 변수에 문자열 할당

```
> var1 <- "data science book"
```

- 변수에 숫자 할당

```
> var2 <- 13
```

- 변수에 날짜 또는 시간 할당

```
> var3 <- Sys.Date()
```

- 값을 여러개의 변수에 연속적으로 할당

```
> var5 <- var4 <- 33
```

- 여러개의 값을 동시에 할당

```
> var6 <- c(1,2,3,4,5)
> var6 <- c(1:5)
```

- 다양한 변수 할당 및 확인

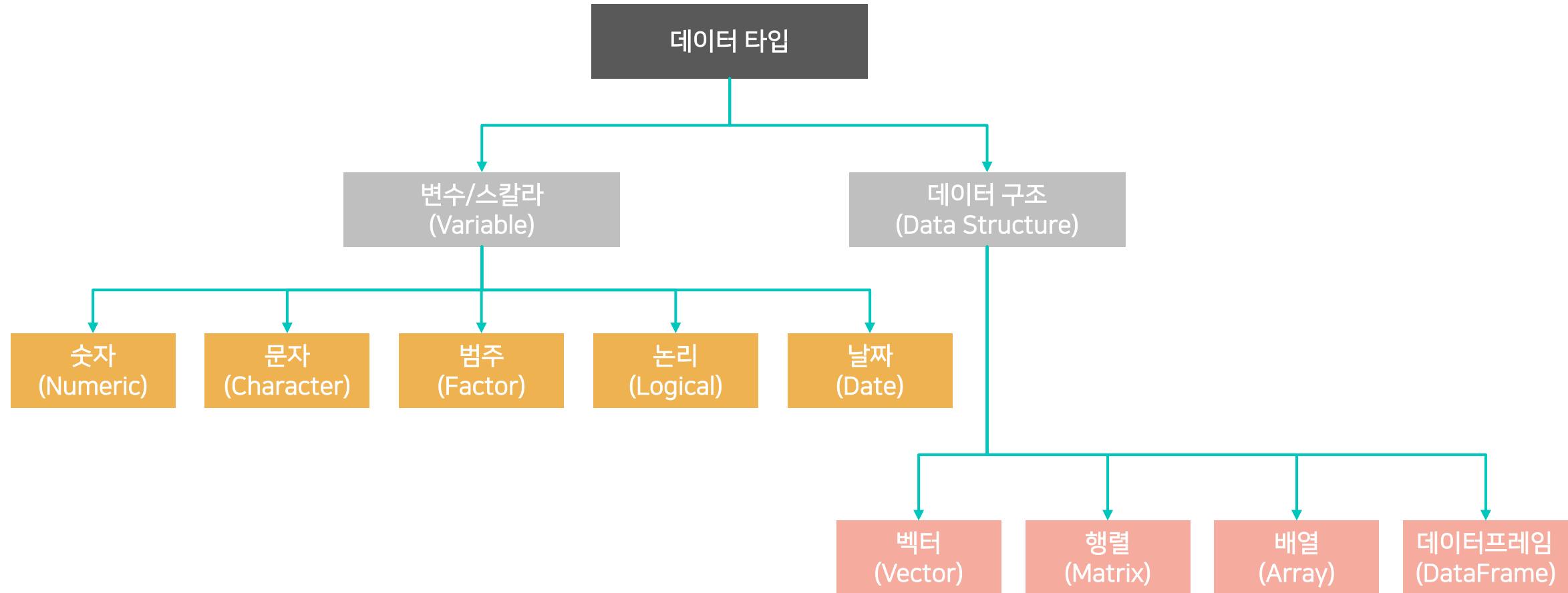
```
> var1 -> 56L; class(var1)
[1] "integer"
> var2 <- 56.7; class(var2)
[1] "numeric"
> var3 <- "dangtong"; class(var3)
[1] "character"
> var4 <- factor(c("continent", "country", "state", "city")); class(var4)
[1] "factor"
> var5 <- as.Date("2019-08-07"); class(var5)
[1] "Date"
> var6 <- as.POSIXct("2019-09-10 09:22:19"); class(var6)
[1] "POSIXct" "POSIXt"
```

정수로 시간 정보를 데이터프레임으로 저장할 때 유용

리스트 형으로 요일, 년, 월, 일 등의 정보를 리스트 내부 원소로 저장되어 유용하다

- 변수 생성 후 관리하기 위한 다양한 명령어가 아래와 같이 존재합니다.

함수	내용
<code>ls()</code>	모든 변수 나열
<code>rm(list=ls())</code>	모든 변수 제거
<code>str()</code>	변수 속성 정보
<code>ls.str()</code>	변수 리스트와 속성을 한번에 보여줌
<code>exists(`var1`)</code>	변수 존재 확인
<code>rm(x) or rm(`x`)</code>	변수 제거
<code>save.image(file="test.RData")</code>	모든 변수를 저장
<code>load("test.RData")</code>	저장된 변수를 불러옴
<code>objects()</code>	생성된 변수 모두 확인하기



- 정수, 부동소수 등의 숫자형 데이터 탑입을 말합니다.

```
> x <- 20  
> y <- 2  
> z <- x * y  
> z  
[1] 40  
  
> print(z)  
[1] 40
```

- R에서는 단 하나의 문자를 표현하는 단위는 없고, 모든 것을 문자열로 사용합니다.

```
string1 <- "문자열입니다"  
string2 <- '문자열 안에 "인용문"이 포함 될때, 저는 작은 따옴표를 사용합니다.'  
  
> "닫는 따옴표가 없는 문자열이다  
+  
+  
+ 도와줘요 같혔어요
```

- 로케일은 ISO 639 코드로 지정된다. 설정하고자 하는 언어의 ISO639 코드를 모르는 경우, [위키피디아](#)에 잘 정리되어 있다. 로케일을 비워 둘 경우에는 운영체제에서 제공한 현재 로케일을 사용합니다.

[https://ko.wikipedia.org/wiki/ISO\\_639-1\\_코드\\_목록](https://ko.wikipedia.org/wiki/ISO_639-1_코드_목록)

- order() 및 sort() 함수는 현재 로케일을 사용하여 정렬한다. 다른 컴퓨터에서도 변함없는 동작을 원한다면 로케일 추가 인수를 취하는 str\_sort() 와 str\_order() 를 사용하면 된다.

```
x <- c("apple", "eggplant", "banana")

str_sort(x, locale = "en") # English
[1] "apple" "banana" "eggplant"

str_sort(x, locale = "haw") # Hawaiian
[1] "apple" "eggplant" "banana"
```

- 범주형(Category) 데이터를 표현하기 위한 타입이며, 아래와 같이 다시 2가지로 나뉩니다.
  - 명목형 (nominal) : 크기나 순서를 비교하기 힘든 범주형 데이터 (갤럭시, 아이폰, 중화폰, 일본폰 등)
  - 순서형 (ordinal) : 크기나 순서가 비교 가능한 데이터 (ex : 20대, 30대, 40대, 50대)

```
x1 <- c("Dec", "Apr", "Jan", "Mar")
x2 <- c("Dec", "Apr", "Jam", "Mar") #-> 글자가 틀려도 상관 없음
```

```
sort(x1)
[1] "Apr" "Dec" "Jan" "Mar" #-> 원하는 순서대로 출력되지 않습니다.
```

```
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct",
"Nov", "Dec")
```

```
y1 <- factor(x1, levels = month_levels)
y1
[1] Dec Apr Jan Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```
sort(y1)
[1] Jan Mar Apr Dec
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- Levels에 지정되지 않은 값이 들어 올 경우 NA로 처리 됩니다.

```
y2 <- factor(x2, levels = month_levels)
y2
[1] Dec Apr <NA> Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- 경고가 발생되길 원하는 경우에는 `readr::parse_factor()`를 사용하면 된다.

```
y2 <- parse_factor(x2, levels = month_levels)
Warning: 1 parsing failure.
row col      expected actual
 3          -- value in level set Jan
```

- Factor 생성과 Level 순서 동시에 (`unique` 함수 사용)

```
f1 <- factor(x1, levels = unique(x1))
f1
[1] Dec Apr Jan Mar
Levels: Dec Apr Jan Mar
```

- 범주형(Category) 데이터를 표현하기 위한 타입이며, 아래와 같이 다시 2가지로 나뉩니다.
  - 명목형 (nominal) : 크기나 순서를 비교 하기 힘든 범주형 데이터
  - 순서형 (ordinal) : 크기나 순서가 비교 가능한 데이터

```
> x <- 20  
> y <- 2  
> z <- x * y  
> z  
[1] 40  
  
> print(z)  
[1] 40
```

- 기본적인 현재 날짜 및 시간 조회

```
library(lubridate)
today()
[1] "2019-01-03"
now()
[1] "2019-01-03 20:32:30 KST"
```

- 문자열에서 생성

```
ymd("2017-01-31")
[1] "2017-01-31"
mdy("January 31st, 2017")
[1] "2017-01-31"
dmy("31-Jan-2017")
[1] "2017-01-31"
```

- 문자열이 아니라도 허용 (ymd 함수 좋아요~~)

```
ymd(20170131)
[1] "2017-01-31"
```

- 데이터 타임형 생성

```
ymd_hms ("2017-01-31 20:11:59")
[1] "2017-01-31 20:11:59 UTC"
mdy_hm ("01/31/2017 08:01")
[1] "2017-01-31 08:01:00 UTC"
```

- 시간대 지정 하여 생성

```
ymd(20170131, tz = "UTC")
[1] "2017-01-31 UTC"
now(tz="Asia/Seoul")
[1] "2019-01-20 22:50:08 KST"
```

# R에서 특수한 성격의 값

값이 존재

값이 존재 X

NA

결측값

(Not Available)

NaN

결측값

(Not A Number)

0/0

`is.na() or is.nan or is.infinite()`

inf

무한

(Infinite)

100/0

-100/0

NULL

# NA/NaN/Inf 확인 및 처리

- 데이트 타임형 생성

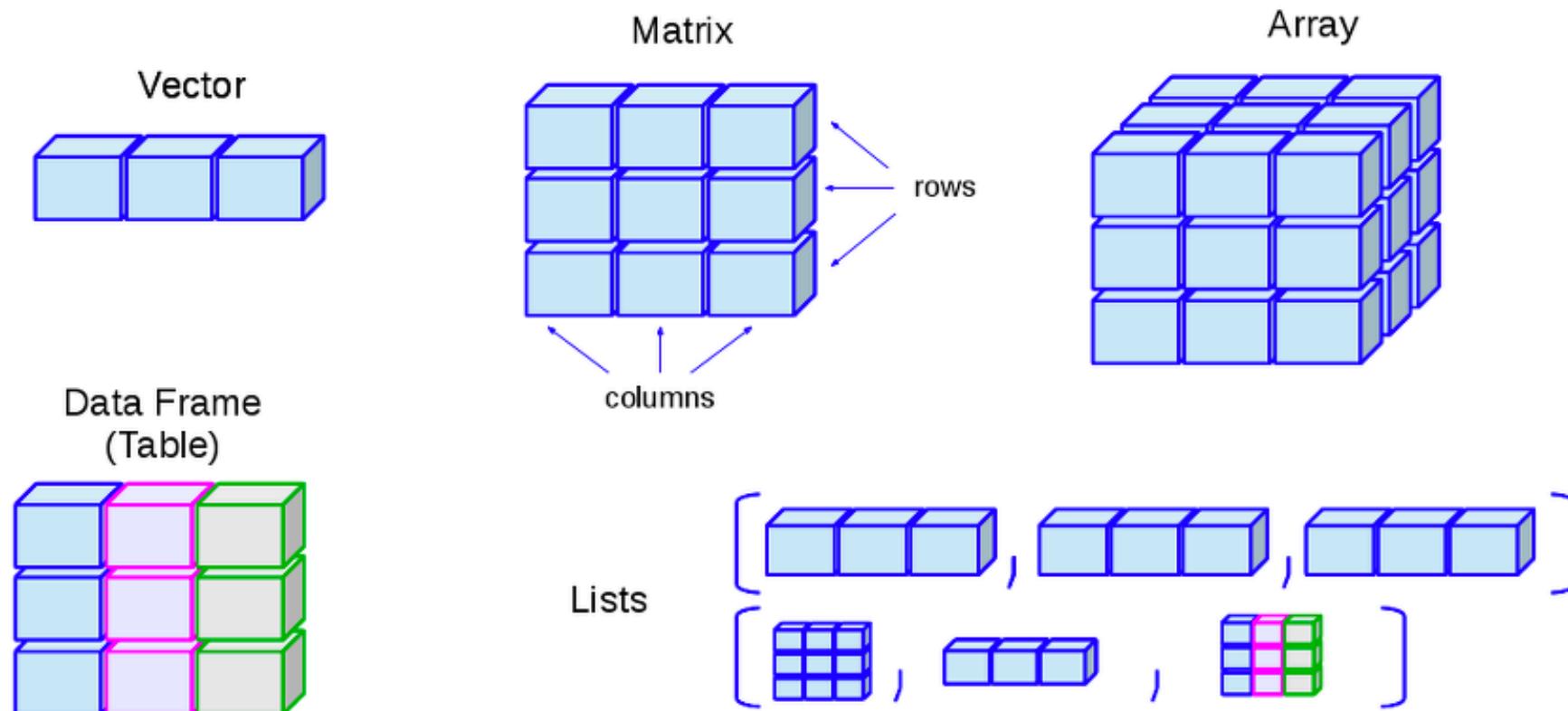
```
z<-c(1,NA,0/0,1/0)
z
[1] 1 NA NaN Inf
is.na(z)
[1] FALSE TRUE TRUE FALSE
is.nan(z)
[1] FALSE FALSE TRUE FALSE
is.finite(z)
[1] TRUE FALSE FALSE FALSE
is.infinite(z)
[1] FALSE FALSE FALSE TRUE
```

- NA 처리 : 대부분의 내장 함수들이 na.rm 매개변수를 수용

```
x <- c(1, 2, NA, NaN)
mean(x, na.rm=TRUE)
[1] 1.5
```

# 데이터 구조

구분	1차원	2차원	N차원
단일형	벡터(vector)	행렬(matrix)	배열(array)
다중형	리스트(list)	데이터프레임(Data Frame)	



- 벡터는 한가지 스칼라 타입의 데이터를 저장 할 수 있습니다.

```
v1 <- c(100, 60, 80, 180)
v2 <- c("A", "B", "C", "D")
```

- 배열의 각 요소에 이름을 부여 할 수 있어서, 색인 뿐만 아니라 이름을 사용해서 접근 가능

```
names(v) <- c("jone", "micheal", "jane", "jessica")
```

- 벡터는 슬라이스를 제공 합니다.

```
> v1[1:3]
jone micheal     jane
100      60       80
```

```
> v1["micheal"]
micheal
  60
> v1[-2]
jone    jane jessica
100      80      180
```

```
> x <- c(1, 3)
> v1[x]
jone jane
100   80
```

- 벡터는 한가지 스칼라 타입의 데이터를 저장 할 수 있습니다.

```
v1 <- c(100, 60, 80, 180)
v2 <- c("A", "B", "C", "D")
```

- 배열의 각 요소에 이름을 부여 할 수 있어서, 색인 뿐만 아니라 이름을 사용해서 접근 가능

```
names(v) <- c("jone", "micheal", "jane", "jessica")
```

- 벡터는 슬라이스를 제공 합니다.

```
> v1[1:3]
jone micheal    jane
100      60      80
```

```
> v1["micheal"]
micheal
  60
> v1[-2]
jone    jane jessica
100      80      180
```

```
> x <- c(1, 3)
> v1[x]
jone jane
100   80
```

- 벡터의 길이 측정

```
length(v1)  
NROW(v1)
```

- 벡터 연산 함수

함수명	기능
identical(x,y)	객체가 동일 한지 판단
union(x,y)	벡터의 합집합
intersect(x,y)	벡터의 교집합
setdiff(x,y)	벡터의 차집합
setequal(x,y)	같은 벡터인지 판단
value %in% x	벡터 x 에 value 가 있는지 판단
x + n	벡터 x 의 모든 요소에 n을 더합니다.(*, %, - 등의 연산자 사용)

- 리스트 : key, value 형태로 데이터 담는 연관 배열
- 리스트 생성 방법

```
list = (  
    key1:value1,  
    key2:value2,  
    ...  
)
```

```
x <- list(name="foo", height=70)  
$name  
[1] "foo"  
  
$height  
[1] 70
```

- 다양한 데이터 탑을 저장

```
x <- list(name="foo", score=c(80,90,100))  
x$name  
[1] "foo"  
  
x$score  
[1] 80 90 100
```

- 리스트 안에 리스트를 중첩하는 것도 가능합니다.

```
list(a=list(val=c(1,2,3)), b=list(val=c(1,2,3,4)))
x$a
$val
[1] 1 2 3

x$b
$val
[1] 1 2 3 4
```

- 리스트 데이터의 접근 : 저장된 색인 또는 키를 사용하여 접근 가능

문법	의미
x\$key	x 리스트에서 key에 해당하는 값
x[n]	x 리스트에서 n번째 데이터의 서브리스트
x[[n]]	x 리스트에서 n번째 저장된 값

- 리스트 안에 리스트를 중첩하는 것도 가능합니다.

```
list(a=list(val=c(1,2,3)), b=list(val=c(1,2,3,4)))
x$a
$val
[1] 1 2 3

x$b
$val
[1] 1 2 3 4
```

- 리스트 데이터의 접근 : 저장된 색인 또는 키를 사용하여 접근 가능

문법	의미
x\$key	x 리스트에서 key에 해당하는 값
x[n]	x 리스트에서 n번째 데이터의 서브리스트
x[[n]]	x 리스트에서 n번째 저장된 값

- 리스트 접근 예

```
x <- list(name="john", score=c(90,100,80))
x$name
[1] "john"

x$score
[1] 90 100 80

x[[1]]
[1] "john"

x[[2]]
[1] 90 100 80

x[1]
$name
[1] "john"

x[2]
$height
[1] 90 100 80
```

- 행렬은 행(row), 열(column)로 만들어진 데이터 구조입니다. 벡터와 마찬가지로 한가지 유형의 스칼라 값만 저장 할 수 있습니다.

```
matrix(  
  data, #행열을 생성할 데이터 벡터  
  nrow, #행의 수  
  ncol, #열의 수  
  byrow=FALSE # TRUE로 설정하면 행 우선, FALSE일 경우 열 우선으로 데이터 채움
```

- 행렬 생성 예

```
matrix(  
  data, #행열을 생성할 데이터 벡터  
  nrow, #행의 수  
  ncol, #열의 수  
  byrow=FALSE # TRUE로 설정하면 행 우선, FALSE일 경우 열 우선으로 데이터 채움
```

# 행렬 생성을 위한 기본 함수

함수	내용
matrix	행렬을 생성
dimnames	객체의 각 차원에 대한 이름을 가져옴
dimnames<-	객체의 각 차원에 이름을 설정
rownames	행렬의 행 이름을 가져옴
rownames<-	행렬의 행 이름을 설정
colnames	행렬의 열 이름을 가져옴
colnames<-	행렬의 열 이름을 설정

- 행렬은 인덱스 또는 행과 열의 이름의 통해 접근 합니다. 벡터의 경우와 마찬가지로 1부터 시작합니다.
  - A행령의 row\_index 행과, col\_index 열의 값에 접근  
 $A[\text{row\_index}, \text{col\_index}]$
  - 만약 row\_index, col\_index 중 하나를 생략하면, 전체 열 또는 전체 행이라고 간주 합니다.

row\_index 생략시

■	■	■	■
■	■	■	■
■	■	■	■
■	■	■	■

col\_index 생략시

■	■	■	■
■	■	■	■
■	■	■	■
■	■	■	■

- 행렬 인덱스로 접근

```
x <- matrix(c(1,2,3,4,5,6,7,8,9), ncol=3)
```

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

```
x[1,1]
```

```
[1] 1
```

```
x[1,2]
```

```
[1] 4
```

```
x[2,1]
```

```
[1] 2
```

```
x[2,2]
```

```
[1] 5
```

- 행렬의 인덱스로 접근

```
x[1:2,]  
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8
```

```
x[-3] # 위와 동일한 결과  
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8
```

```
x[c(1,3),c(1,3)] #인덱스를 벡터로 지정  
      [,1] [,2]  
[1,]    1    7  
[2,]    3    9
```

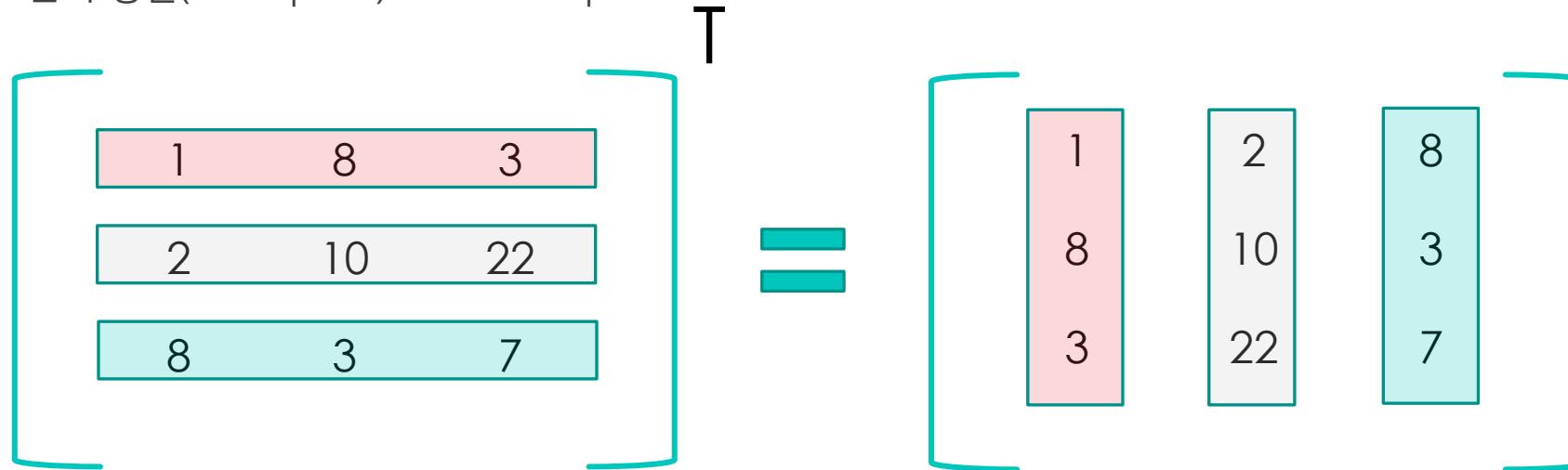
- 행렬 연산자

행렬 연산	설명
$A + x$	$A$ 행렬의 모든 값에 $x$ 를 더함. $-$ , $*$ , $/$ 연산도 가능
$A + B$	행렬 $A$ 와 행렬 $B$ 의 합. 차는 – 연산자 이용
$A \%*% B$	행렬 $A$ 와 행렬 $B$ 의 곱

- 행렬 연산 함수

행렬 연산	설명
$t$	행렬 또는 데이터 프레임의 전치 행렬
$solve$	수식 $a \%*% x = b$ 에서 $x$ 를 구합니다. (행렬 방정식의 해를 구하는 것)
$nrow$	배열의 행의 수를 구함
$ncol$	배열의 열의 수를 구함
$dim$	객체의 차원 수를 구하거나 지정

- 전치 행렬(Transpose) – 2D Transpose



- 왜 쓸까요?

$$y = ax_1 + bx_2 + cx_3 + dx_4 + ex_5$$

$\begin{bmatrix} 1 & 8 & 3 & 4 & 1 \\ 2 & 10 & 22 & 7 & 2 \\ 8 & 3 & 7 & 6 & 34 \end{bmatrix} \times \begin{bmatrix} a & b & c & d & e \end{bmatrix}$

행렬 연산 불가

$\begin{bmatrix} 1 & 2 & 8 \\ 8 & 10 & 3 \\ 3 & 22 & 7 \\ 4 & 7 & 6 \\ 1 & 2 & 34 \end{bmatrix} \times \begin{bmatrix} a & b & c & d & e \end{bmatrix}$

연산 가능

## ▪ 전치 행렬의 성질

$$(1) \quad (A^T)^T = A$$

$$(2) \quad (A + B)^T = A^T + B^T$$

$$(3) \quad (AB)^T = B^T A^T$$

$$(4) \quad (kA)^T = kA^T$$

- 행렬이 2차원 데이터라면 배열(Array)는 다차원 데이터입니다. 예를 들어  $4 \times 5$  차원의 데이터를 행렬로 표현한다면  $2 \times 3 \times 4$  차원의 데이터는 배열로 표현 합니다.
- 배열의 생성

```
array(  
  data = NA,          #데이터를 저장한 벡터  
  dim=length(data),  #배열의 차원. 이 값을 지정하지 않으면 1차원 배열이 생성 됩니다.  
  dimnames,           #차원의 이름  
)
```

```
array(1:12, dim=c(3,4))  
  
[,1] [,2] [,3] [,4]  
[1,]    1    4    7   10  
[2,]    2    5    8   11  
[3,]    3    6    9   12
```

# 다차원 배열 만들어 보기

## 02. 데이터 탑 및 구조

- 4 X 2 X 3 차원의 배열을 만들어 봅니다.

```
x <- array(1:12, dim=c(4,2,3))  
x  
, , 1  
[,1] [,2]  
[1,] 1 5  
[2,] 2 6  
[3,] 3 7  
[4,] 4 8  
  
, , 2  
[,1] [,2]  
[1,] 9 1  
[2,] 10 2  
[3,] 11 3  
[4,] 12 4  
  
, , 3  
[,1] [,2]  
[1,] 5 9  
[2,] 6 10  
[3,] 7 11  
[4,] 8 12
```

```
x[,,3]  
[,1] [,2]  
[1,] 5 9  
[2,] 6 10  
[3,] 7 11  
[4,] 8 12  
  
x[1,,3]  
[1] 5 9  
  
x[,2,3]  
[1] 9 10 11 12  
  
x[2,2,]  
[1] 6 2 10  
  
dim(x)  
[1] 4 2 3
```

# 데이터 프레임(Data Frame)

- R에서 가장 많이 쓰는 데이터 구조이며, 테이블형 데이터셋 (Tabular Dataset) 이라고도 합니다.
- 각 열에는 관측 값의 이름이 저장되고, 각 행에는 매 관측 단위마다 실제 얻어진 값이 저장 됩니다.

자동차	주행거리	최고시속	구입년도
BMW	120,000	240	2010
Lexus	1,200	250	2019
Tesla	300,000	300	2008
Hyundai	124,000	220	2011

- 데이터 프레임 생성

```
data.frame(변수명1, 변수명2, ..., 변수명 n)
```

```
ID <- c("BMW", "Lexus", "Tesla", "Hyundai")
DISTANCE <- c(120000, 1200, 300000, 124000)
SPEED <- c(240, 250, 300, 220)
ORDER_YEAR <- c(2010, 2019, 2008, 2011)
```

```
dataframe_test <- data.frame(ID, DISTANCE, SPEED, ORDER_YEAR)
dataframe_test
```

	ID	DISTANCE	SPEED	ORDER_YEAR
1	BMW	120000	240	2010
2	Lexus	1200	250	2019
3	Tesla	300000	300	2008
4	Hyundai	124000	220	2011

- 데이터 프레임의 데이터를 조회하고 관측 값을 수정해봅니다.

```
#데이터 프레임 생성
```

```
x <- c(1,2,3,4,5)
y <- c(2,4,6,8,10)
z <- c('M','F','M','F','M')
d <- data.frame(x,y,z)
d
```

	x	y	z
1	1	2	M
2	2	4	F
3	3	6	M
4	4	8	F
5	5	10	M

```
#x열만 조회
```

```
d$x
[1] 1 2 3 4 5
```

```
#q열 추가
```

```
d$q <- 6:10
x y z q
1 1 2 M 6
2 2 4 F 7
3 3 6 M 8
4 4 8 F 9
5 5 10 M 10
```

```
#q열을 설정
```

```
d$q <- 20:24
x y z q
1 1 2 M 20
2 2 4 F 21
3 3 6 M 22
4 4 8 F 23
5 5 10 M 24
```

```
#q열의 개수를 초과한 데이터 입력
```

```
d$q <- 30:35
Error in `$<- .data.frame`(`*tmp*`, q, value = 30:35) :
replacement has 6 rows, data has 5
```

```
d[3,2]
```

```
[1] 6
```

```
d[-3,2]
```

```
[1] 2 4 8 10
```

```
d[3]
```

```
z
1 M
2 F
3 M
4 F
5 M
```

```
d[,c("x","y")]
```

```
x y
1 1 2
2 2 4
3 3 6
4 4 8
5 5 10
```

- 행과 열의 이름을 조회 합니다.

함수	설명
colnames(x)	컬럼명을 조회 합니다.
rownames(x)	열명을 조회 합니다.
str(x)	데이터 프레임 속성 조회

- 유틸리티 함수

함수	설명
head(x, n=6L)	앞부분 부터 n 만큼 잘라낸 데이터 표시
tail(x, n=10L)	뒷부분 부터 n 만큼 잘라낸 데이터 표시
View(x)	데이터 뷰어를 호출 합니다.

- 여러 함수를 호출 하다 보면 반환 되는 값이 어떤 데이터 타입인지 분명하지 않을 때가 있습니다. 이러한 경우 다음 함수들을 사용하여 데이터 타입을 손쉽게 판단할 수 있습니다.

함수	설명
class()	객체 x의 클래스
str()	객체 X의 속성
is.factor(x)	팩터 확인
is.numeric(x)	숫자를 저장한 벡터인가
is.character(x)	문자열을 저장한 벡터인가
is.matrix(x)	주어진 객체 X가 행렬인가
is.array(x)	주어진 객체 X가 배열인가
is.data.frame(x)	주어진 객체 X가 데이터 프레임 인가

- 행과 열의 이름을 조회 합니다.

함수	설명
as.factor(x)	객체 x 를 팩터로 변환
as.numeric(x)	객체 x 를 숫자를 저장한 벡터로 변환
as.character(x)	객체 x 를 문자열을 저장한 벡터로 변환
as.matrix(x)	객체 x 를 행렬로 변환
as.array(x)	객체 x 를 배열로 변환
as.data.frame(x)	객체 x 를 데이터 프레임으로 변환

# 3장. 흐름제어 및 연산

There are only hard things in Computer Science.

Cache Invalidation and naming things

- if문

```
if (condition) {  
    condition 이 "참" 일 경우 실행 할 문장  
} else {  
    condition 이 "거짓" 일 경우 실행 할 문장  
}
```

- if문 예제

```
if (3 < 10) {  
    print("TRUE")  
    print("3은 10보다 작습니다.")  
} else {  
    print("FALSE")  
    print("설마 3이 10보다 클까요?")  
}
```

- "참"일 때와 "거짓"일 때 리턴 되는 각각의 값을 Inline 으로 지정 가능

```
ifelse {  
  <test expression>,  
  <"참" 일 경우 수행 문장>,  
  <"거짓" 일 경우 수행 문장>  
}
```

- ifelse문 예제

```
x1 <- c(10,11,12,13,14)  
ifelse ( x1 %% 2 == 0, "짝수", "홀수")  
[1] "짝수" "홀수" "짝수" "홀수" "짝수"
```

- 여러 개의 조건식이 필요 할 경우 사용

```
if (exp1) {  
    <exp1 "참" 일 경우 수행 문장>  
} else if (exp2) {  
    <exp2 "참" 일 경우 수행 문장>  
} else {  
    <exp1 및 exp2 모두 "거짓" 일 경우 수행 문장>  
}
```

## 실습 교제 참고

- 일정한 횟수 만큼 반복이 필요 한 경우 사용

```
for (변수 in 횟수) {  
    <수행 코드>  
}
```

- for 문 예제

```
> for (i in 1:10) {  
print(i)  
}  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10
```

- 조건 표현식이 "참" 인 동안 반복 수행

```
while (exp) {  
    <수행 코드>  
}
```

- while 문 예제

```
> i <- 1  
> while (i <= 10) {  
print(i)  
i <- i + 1  
}  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10
```

- 반복문 수행시 next 를 만나면 현재 수행중인 반복문 을 중단하고, 다음 반복을 다시 시작 합니다.

```
while (exp) {  
    <수행 코드>  
}
```

- while 문 예제 : 1 부터 10까지 짝수만 출력

```
> i <- 1  
> while (i <= 10) {  
    i <- i + 1  
    if(i %% 2 !=0) {  
        next  
    }  
    print(i)  
}  
[1] 2  
[1] 4  
[1] 6  
[1] 8  
[1] 10
```

```
> i <- 1  
> while (i <= 10) {  
    if(i %% 2 !=0) {  
        next  
    }  
    i <- i + 1  
    print(i)  
}  
무한 루프!!!!
```

- repeat 문 : 무조건 반복만 수행 함으로 항상 break 문과 함께 사용 해야함.
- break 문 : 반복을 종료

```
repeat {  
  <수행 코드>  
  if (exp) {  
    break  
  }  
}
```

- while 문 예제 : 1 부터 10까지 짹수만 출력

```
> i <- 1  
> repeat {  
  print(i)  
  if(i >= 5) {  
    break  
  }  
  i <- i + 1  
}  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5
```

연산자	내용
$+, -, *, /$	더하기, 빼기, 곱하기, 나누기
$\wedge$ 또는 $**$	거듭제곱
$<$	미만
$\leq$	이하
$>$	초과
$\geq$	이상
$= =$	상등
$\neq$	부등
$!x$	부정(참이면 거짓, 거짓이면 참)
$x   y$	논리합
$x \& y$	논리곱
<code>isTRUE(x)</code>	$x$ 가 참인지 확인

연산자	내용
$n \% \% m$	$n$ 을 $m$ 으로 나눈 나머지
$n \% / \% m$	$n$ 을 $m$ 으로 나누 몫
$n^m$	$n$ 의 $m$ 승
$\exp(n)$	지수함수
$\log(x, \text{base}=\exp(1))$	로그함수
$\sin(x), \cos(x), \tan(x)$	삼각함수
$\sqrt{x}$	제곱근
$\max(), \text{pmax}()$	최대값
$\min(), \text{pmin}()$	최소값
$\sum$	합
$\text{mean}$	평균
$\text{abs}$	절대값
$\text{cummax}, \text{cummin}, \text{cumprod}, \text{cumsum}$	누적연산
$\text{ceiling}, \text{round}, \text{trunc}, \text{floor}$	올림, 반올림...

# 4장. 함수

There are only hard things in Computer Science.

Cache Invalidation and naming things

- 함수를 이용하면 반복된 작업을 모듈화 하여 가독성을 높이고 재사용 성을 증대 시킬 수 있습니다.
  - 함수 기본 정의

```
function_name <- function(param1, param2, ...) {  
    <함수 내용>  
    return(<리턴값>)  
}
```

- 가변 길이 함수 : 함수 파라메터의 갯수가 정해 지지 않을 경우에 사용

```
> func<- function(...) {  
    args <- list(...)  
    for( x in args) {  
        print(x)  
    }  
}  
> func("a", "b")  
[1] "a"  
[1] "b"
```

- 함수 안에 함수를 둘 수 있습니다. 주로 함수 안에서 재사용 성을 높이기 위해 사용 합니다.

```
> func <- function(x, y) {  
  print(x)  
  g <- function(y) {  
    print(y)  
  }  
  g(y)  
}  
> f(1,2)  
[1] 1  
[1] 2
```

- 함수의 생명주기가 종료 됐지만, 내부 함수가 참조 하고 있어서 그 함수에 접근할 수 있는 함수
- 외부 함수의 변수는 내부 익명 함수의 Scope에 의해 보호 됩니다.
- 익명 함수를 사용합니다.

```
func1 <- function(a,b) {  
  outer = a + b  
  return(  
    func2(c,d) {  
      inner = c + d  
    }  
  )  
}
```

```
click_counter <- function() {  
  count = 0  
  return(  
    function(x) {  
      count <-- count + x  
      print(count)  
    }  
  )  
}  
  
counter = func1()  
counter(1)  
counter(1)  
  
> counter = func1()  
> counter(1)  
[1] 1  
> counter(1)  
[1] 2
```

- 함수 파라미터는 default 값을 지정 할 수도 있고, 값을 주지 않으면 default 값다

```
func <- function(x, y=1, z=10) {  
  return(x+y+z)  
}  
  
> func(1)  
[1] 12  
> func(1,2)  
[1] 13  
> func(1,2,3)  
[1] 6  
>
```

- 스코프란
  - 코드에 사용한 변수의 범위를 지정하는 규칙
  - 변수가 정의된 블록 내에서만 변수 사용 가능
- 전역 변수 및 지역 변수
  - 전역 변수는 모든 블록에서 사용 가능한 변수
  - 전역 변수는 실행 중인 현재 세션에서만 유효
  - 콘솔에서 변수를 선언하면 전역 변수로 생성 됩니다.

- 콘솔에서 정의된 전역변수는 현재 세션에서는 재사용 가능

```
n <- 1
func <- function() {
  print(n)
}
func()
[1] 1

n <- 2
func()
[1] 2
```

- 같은 이름으로 생성된 변수가 있다면 함수 내부에서 정의된 변수가 우선 순위.
- 상위 변수를 바꾸고 싶으면 <- 대신 <<- 를 사용하면 됨.

```
n <- 100
f <- function() {
  n <- 1
  print(n)
}
f()
[1] 1
```

# 4장. 입출력

There are only hard things in Computer Science.

Cache Invalidation and naming things

항목	base	data.table	readr	base	feather
	write.csv() read.csv	fwrite() fread()	write_csv() read_csv()	saveRDS() readRDS()	write_feather() read_feather()
숫자 읽기	33.60s	10.25s	3.18s	0.87s	0.26s
숫자 쓰기	28.50s	0.28s	2.96s	13.12s	0.07s
문자 읽기	6.29s	0.54s	0.57s	1.30s	0.27s
문자 쓰기	6.37s	0.08s	2.71s	6.76s	0.80s

- 작업 디렉토리 지정
  - `setwd("c://bigdata//data")` : 현재 작업 디렉토리 설정
  - `getwd()` : 현재 작업 디렉토리 조회
- 또는 아래와 같이 "/" 를 사용해서 지정도 가능
  - `setwd("c:/bigdata/data")`

- \*.xls, \*.xlsx 파일을 데이터 세트로 변환

```
install.packages("readxl")      # 패키지 설치
library(readxl)                  # 패키지 로드
getwd()

excel_data_ex <- read_excel("data_ex.xls", sheet=1)

## # A tibble: 6 x 4
## ID SEX AGE AREA
## <dbl> <chr> <dbl> <chr>
## 1 1 F 22 서울
## 2 2 F 12 부산
## 3 3 F 34 마산
## 4 4 F 44 청주
## 5 5 M 12 서울
## 6 6 F 34 청도

View(excel_data_ex)
```

- 탭으로 구분된 TXT 파일 가져오기

```
ex_data <- read.table("data.ex.txt")
View(ex_data)

# 첫번째 행을 값으로 인식 하지 말고 변수명 으로 인식 하기
ex_data <- read.table("data_ex.txt", header = TRUE)
View(ex_data)
```

- 옵션 설명

- header : 1행이 변수명 인지 아닌지 판단(1행이 변수명이면 header = TRUE)
- skip : 특정 행까지 제외하고 데이터 가져오기 (4행부터 가져오려면 skip = 3)
- nrows : 특정 행까지 데이터 가져오기 (10행까지 가져오려면 nrows = 7)
- sep : 데이터 구분자 지정(쉼표로 구분된 데이터가 있다면 sep = ",")