Accessing the help files

?mean
Get help of a particular function.

help.search('weighted mean')
Search the help files for a word or phrase.

help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.

class(iris)
Find the class an object belongs to.

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Input	Ouput
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`
`load('file.RData')`	`save(df, file = 'file.Rdata')`

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.

rev(x)
Return x reversed.

table(x)
See counts of values.

unique(x)
See unique values.

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Matrices

```
m <- matrix(x, nrow = 3, ncol = 3)
```
Create a matrix from x.


`m[2, ]` - Select a row


`m[ , 1]` - Select a column


`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
```
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

Also see the **dplyr** package.

Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` 

`df[[2]]` 

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]` 

`df[2, ]` 

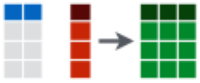
`df[2, 2]` 

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

cbind - Bind columns.



rbind - Bind rows.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Strings

```
paste(x, y, sep = ' ')
paste(x, collapse = ' ')
grep(pattern, x)
gsub(pattern, replace, x)
toupper(x)
tolower(x)
nchar(x)
```

Conditions

a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

For Loop

```
for (variable in sequence){
    Do something
}
```

Example

```
for (i in 1:4){
    j <- i + 10
    print(j)
}
```

While Loop

```
while (condition){
    Do something
}
```

Example

```
while (i < 5){
    print(i)
    i <- i + 1
}
```

If Statements

```
if (condition){
    Do something
} else {
    Do something different
}
```

Example

```
if (i > 3){
    print('Yes')
} else {
    print('No')
}
```

Functions

```
function_name <- function(var){
    Do something
    return(new_variable)
}
```

Example

```
square <- function(x){

    squared <- x*x

    return(squared)
}
```

[정규표현식 : 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식 언어]

```
word <- "JAVA javascript 가나다 123 %^&*"

gsub("A", "", word)

gsub("a", "", word)

gsub("[Aa]", "", word)

gsub("[가-힣]", "", word)

gsub("[^가-힣]", "", word)

gsub("[&^%*]", "", word)

gsub("[[:punct:]]", "", word)

gsub("[[:alnum:]]", "", word)

gsub("[1234567890]", "", word)

gsub("[[:digit:]]", "", word)

gsub("[^[:alnum:]]", "", word)

gsub("[[:space:]]", "", word)

gsub("[[:space:][:punct:]]", "", word)
```

.	Any character except \n
|	Or, e.g. (a|b)
[...]	List permitted characters, e.g. [abc]
[a-z]	Specify character ranges
[^...]	List excluded characters
(...)	Grouping, enables back referencing using \\N where N is an integer

[[:digit:]] or \\d	Digits; [0-9]
\\D	Non-digits; [^0-9]
[[:lower:]]	Lower-case letters; [a-z]
[[:upper:]]	Upper-case letters; [A-Z]
[[:alpha:]]	Alphabetic characters; [A-z]
[[:alnum:]]	Alphanumeric characters [A-z0-9]
\\w	Word characters; [A-z0-9_]
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; [0-9A-Fa-f]
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; [^[:space:]]
[[:punct:]]	Punctuation characters; !"#$%&'()*+,-./:;<=>?@[]^_`{|}~
[[:graph:]]	Graphical char.; [[:alnum:][:punct:]]
[[:print:]]	Printable characters; [[:alnum:][:punct:]\\s]
[[:cntrl:]] or \\c	Control characters; \n, \r etc.

[문자열 처리 함수들]

nchar()
sort()
tolower()
toupper()
substr()
substring()
gsub()
grep()
strsplit()

[날짜와 시간관련 함수들]

- 현재날짜: Sys.Date()
- 현재날짜 및 시간: Sys.time()
- 미국식 날짜 및 시간: date()

- 년월일 시분초 타입의 문자열을 날짜 또는 시간으로 변경 :
  as.Date("년-월-일 시:분:초") 또는 as.Date("년/월/일 시:분:초")
  as.POSIXct("년-월-일 시:분:초") 또는 as.POSIXct("년/월/일 시:분:초")
  as.POSIXlt("년-월-일 시:분:초") 또는 as.POSIXlt("년/월/일 시:분:초")

- 특정 포맷을 이용한 날짜: as.Date("날짜 문자열", format="포맷")
  as.POSIXct("날짜와 시간 문자열", format="포맷")
  as.POSIXlt("날짜와 시간 문자열", format="포맷")

- 날짜 데이터끼리 연산 가능 :
  날짜끼리 뺄셈가능, 날짜와 정수의 덧셈뺄셈 가능 - 하루를 1로 간주, 소숫점 생략
  날짜 데이터끼리 연산할 때 소숫점을 표현하고자 하는 경우는 as.Date 대신에 as.POSIXct 함수를 이용

```
today <- Sys.Date()
format(today, "%d %B %Y")
weekdays(today); months(today); quarters(today)
unclass(today)   # 1970-01-01을 기준으로 얼마나 날짜가 지났는지의 값을 가지고 있다.
Sys.Date();Sys.time()
Sys.timezone()
```

Symbol	Meaning	Example
%d	day as a number (1-31)	01-31
%a	abbreviated weekday	Mon

```
as.Date('1/15/2018',format='%m/%d/%Y')
as.Date('4월 26, 2018',format='%B %d, %Y')
as.Date('22118',format='%d%b%y')

x1 <- "2019-01-10"
# 문자열을 날자형으로
as.Date(x1, "%Y-%m-%d")
# 문자열을 날짜+시간형으로
strptime(x1, "%Y-%m-%d")
x2 <- "20180601"
as.Date(x2, "%Y%m%d")
strptime(x2, "%Y%m%d")

as.Date("2022/01/01 08:00:00") - as.Date("2022/01/01 05:00:00")
as.POSIXct("2022/01/01 08:00:00") - as.POSIXct("2022/01/01 05:00:00")
as.POSIXlt("2022/01/01 08:00:00") - as.POSIXlt("2022/01/01 05:00:00")

t<-Sys.time()
ct<-as.POSIXct(t);
lt<-as.POSIXlt(t)
unclass(ct);
unclass(lt); lt$mon; lt$hour; lt$year+1900

as.POSIXct(1449894437,origin="1970-01-01")
as.POSIXlt(1449894437,origin="1970-01-01")
```

[apply 계열 함수]

R에는 벡터, 행렬 또는 데이터 프레임에 임의의 함수를 적용한 결과를 얻기 위한 apply 계열 함수가 있다. 이 함수들은 데이터 전체에 함수를 한 번에 적용하는 벡터 연산을 수행하므로 속도도 빠르고 구현도 간단하다.

함수	설명	다른 함수와 비교했을 때의 특징
apply( )	배열 또는 행렬에 주어진 함수를 적용한 뒤 그 결과를 벡터, 배열 또는 리스트로 반환	배열 또는 행렬에 적용
lapply( )	벡터, 리스트 또는 표현식에 함수를 적용하여 그 결과를 리스트로 반환	결과가 리스트
sapply( )	lapply 와 유사하나 결과를 가능한 심플한 데이터셋으로 반환	결과가 심플데이터셋

tapply( )	벡터에 있는 데이터를 특정 기준에 따라 그룹으로 묶은 뒤 각 그룹마다 주어진 함수를 적용하고 그 결과를 반환	데이터를 그룹으로 묶은 뒤 함수를 적용
mapply( )	sapply 의 확장된 버전으로, 여러 개의 벡터 또는 리스트를 인자로 받아 함수에 각 데이터의 첫째 요소들을 적용한 결과, 둘째 요소들을 적용한 결과, 셋째 요소들을 적용한 결과 등을 반환	여러 데이터셋의 데이터를 함수의 인자로 적용한 결과

[apply()]

apply( )는 행렬의 행 또는 열 방향으로 특정 함수를 적용하는 데 사용한다.

apply : 배열 또는 행렬에 함수 FUN을 MARGIN 방향으로 적용하여 결과를 벡터, 배열 또는 리스트로 반환한다.

```
apply(
  X,        # 배열 또는 행렬
  MARGIN,   # 함수를 적용하는 방향. 1은 행 방향, 2는 열 방향
  FUN       # 적용할 함수
)
```

반환 값은 FUN이 길이 1인 벡터들을 반환한 경우 벡터, 1보다 큰 벡터들을 반환한 경우 행렬, 서로 다른 길이의 벡터를 반환한 경우 리스트다. apply( )가 적용된 결과가 벡터, 배열, 리스트 중 어떤 형태로 반환될 것인지는 데이터 X의 데이터 타입과 함수 FUN의 반환 값에 따라 대부분 자연스럽게 예상할 수 있다. 다음은 합을 구하는 함수 sum( )을 apply( )에 적용하는 예이다.

```
> d <- matrix(1:9, ncol=3)
> d
> apply(d, 1, sum)
[1] 12 15 18
> apply(d, 2, sum)
[1] 6 15 24
```