

## R02. R Basics for the first-timers

Sim, Min Kyu, Ph.D.  
mksim@seoultech.ac.kr



서울과학기술대학교 데이터사이언스학과

- 1 Data Type
- 2 Data Structure
- 3 Control Statement
- 4 Base Cheatsheet

# Data Type

# Motivation

## 주민등록번호

The diagram illustrates the components of a Korean Resident Registration Number (주민등록번호) and how they are stored in a database table. The number '920708-1234567' is shown with boxes around each part: '손흥민' (Name), '920708' (Birth Year/Month/Day), '1' (Gender), '2345' (Residence), '6' (Check Digit), and '7' (Validity Check). Lines connect these components to the corresponding columns in the table below.

	손흥민	920708	1	2345	6	7
정보	이름	생년월일	1900년대 출생 남성	출생고유지 등록번호	해당 출생지 일련번호	Validity Check
특성	문자	날짜	분류	분류	숫자	숫자
	Character	Date	Category	Category	Number	Number
R Data Type	character	Date	factor	factor	numeric	numeric

- 하나의 문자열로 보이는 주민등록번호에 의미적으로 여러가지 data type이 담겨있다.
- 주민등록번호는 “숫자”가 아니다. (“숫자”는 사칙연산이 가능해야 한다.)
- Data Type(문자, 날짜, 분류, 숫자)에 따라 다루는 방법이 다르다.

## R Data Types

1. character (string)
2. numeric
3. logical
4. factor
5. Date
6. and many others...

# 1. *character* (string)

```
greeting <- "R says \"Hello World!\""
nchar(greeting) # number of characters
```

```
## [1] 21
```

```
substr(greeting, 3, 6) # substring from 3rd to 6th
```

```
## [1] "says"
```

```
greeting # show
```

```
## [1] "R says \"Hello World!\""
cat(greeting) # show cleanly
```

```
## R says "Hello World!"
```

- Random tips

- 따옴표를 입력할때는 backslash를 앞에 붙여준다.
- substr은 SUBset of STRing, 문자열의 부분 집합을 추출한다.
- cat함수를 사용하면 backslash를 빼고 출력한다.

```
paste0(string1, string2); paste(string1, string2, sep)
nchar(string); substr(string, start, end); cat(string)
```

Strings		Also see the <b>stringr</b> package.
<code>paste(x, y, sep = ' ')</code>	Join multiple vectors together.	
<code>paste(x, collapse = ' ')</code>	Join elements of a vector together.	
<code>grep(pattern, x)</code>	Find regular expression matches in x.	
<code>gsub(pattern, replace, x)</code>	Replace matches in x with a string.	
<code>toupper(x)</code>	Convert to uppercase.	
<code>tolower(x)</code>	Convert to lowercase.	
<code>nchar(x)</code>	Number of characters in a string.	

그림 1: base 패키지에서 제공하는 string 처리 함수

- base 패키지의 함수들 외에도
- stringr, stringi의 패키지가 string 처리 함수들을 제공한다.
- tidyverse 패키지는 stringr, stringi 패키지를 포함한다.

## 2. numeric

```
10^2 + 36
```

```
## [1] 136
```

```
a <- 4
```

```
a
```

```
## [1] 4
```

```
a*5
```

```
## [1] 20
```

```
a <- a + 10 # assign a+10 to a
```

```
a
```

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

- 일반 계산기 처럼 사용할 수 있다.



### 3. logical

- 참과 거짓 (TRUE와 FALSE)
- TRUE는 수치형의 1, FALSE는 수치형의 0에 대응된다.

```
2==3
```

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

```
5>3
```

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

```
as.numeric(2==3)
```

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

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

그림 2: logical 값을 반환하는 수치 비교 함수

## Data Type의 확인과 변환

- `is.DATATYPE()` 함수
  - 해당 DATATYPE이 맞으면 TRUE, 아니면 FALSE값을 반환한다.
- `as.DATATYPE()` 함수
  - 인수를 DATATYPE으로 변환하여 반환한다.

```
is.character(5)
```

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

```
is.character("5")
```

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

```
a <- as.character(5)
```

```
is.character(a)
```

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

```
b <- as.numeric(a)
```

```
is.numeric(b)
```

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

```
as.numeric(2==3)
```

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

- `class()` 함수

- 인수의 Data Type을 반환한다.

```
class(5)
```

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

```
class("TRUE")
```

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

```
class(TRUE)
```

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

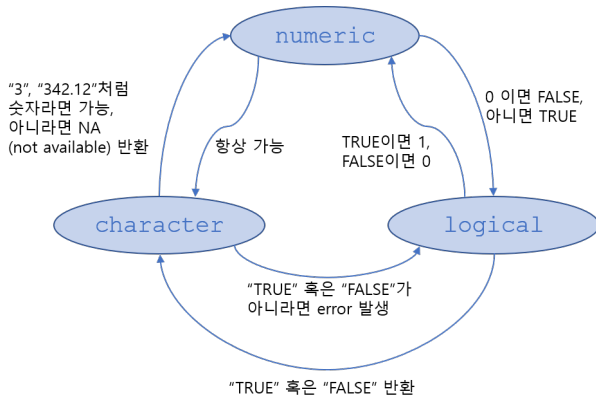


그림 3: string-numeric-logical간의 변환

### ● 함수들 마다 제각각

- 입력(input)에 해당하는 인수(argument)의 지정된 data type이 있다.
- 출력(output)값도 지정된 data type으로 반환된다.
- 그러므로 data type을 확인하고 변환할 수 있어야 한다.

Types		
Converting between common data types in R. Can always go from a higher value in the table to a lower value.		
<code>as.logical</code>	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
<code>as.numeric</code>	1, 0, 1	Integers or floating point numbers.
<code>as.character</code>	'1', '0', '1'	Character strings. Generally preferred to factors.
<code>as.factor</code>	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

그림 4: Base Cheatsheet의 data type 관련 부분

- 위에서 아래 방향으로는 언제나 변환이 가능하다.
- 데이터 타입의 개수는 매우 많다.
- 새로운 응용프로그램 (패키지)마다 적합한 타입을 정의하기도 한다.
- 마주칠때마다 검색하고 필요한 만큼 알아보고 사용한다.

## 4. *factor* (Categorical, 범주형 변수)

### Group을 나타내는 data type

- Keywords
  - “Partition”, “Classification”, “분류”, “집단”, “Group”, “범주”
- 예시
  - 성인남자, 성인여자, 미성년자
  - 표준 산업 분류 (제조업, 금융업, 광공업등)
  - 날씨 (맑음, 흐림, 비가 올)
- 생성
  - `data.frame` 생성시에 `stringsAsFactors = TRUE`를 사용하면 모든 문자 객체가 `factor`가 됨
  - 다른 `data type`에서 `as.factor()`를 이용해서 변환

### 다른 data type과의 차이

- 숫자 vs categorical
  - 더하고 뺄 수 있다면 `numeric`
  - 1,2,3을 A,B,C로 바뀌어도 무리가 없다면 `categorical`
- 문자 vs categorical
  - Exclusive (배타적) 집합이고, 각 개체가 1개의 그룹에 속한다면 `categorical`

# Data Structure

## 자료형 (data type) vs 자료구조 (data structure)

### Data type

- 변수에 입력된 하나의 값의 특성
- 0차원, 하나의 점, 하나의 값, singleton

### Data Structure

- 묶어서 보관/처리하기 위해서 필요
- 각각의 값(singleton)들이 모여 있는 구조
- 대용량 데이터도 한 번에 포함할 수 있기에 데이터 분석에서 중요
- 엑셀에서 1개의 컬럼, 1개의 네모 블록, 1개의 워크시트, 1개의 파일 모두 자료 구조에 해당

### 자료 구조의 이해

1. 길다란가? 네모난가? 뽀돌뽀돌한다?
2. 몇 개의 관찰값이 있는가?
3. 어떤 규칙을 가지고 있는가?



## Data Structure의 종류

1. `vector`
2. `array (matrix)`
3. `data.frame`
4. `list`

# 1. *vector*

- 길다랗게 저장되어 있는 데이터 구조
- `c()` 함수를 이용해서 벡터를 생성

## Character vectors

- `paste` 함수는 string으로 된 vector에도 적용이 가능하다.
- `seq()` 함수는 등차 수열을 만든다.
- `a:b`는 a부터 b까지의 정수 벡터를 만든다.

```
strVec1 <- c("Hello", "Hi", "What's up")
```

```
strVec1
```

```
## [1] "Hello"      "Hi"         "What's up"
```

```
strVec2 <- c("Ma'am", "Sir", "Your Honor")
```

```
strVec3 <- paste(strVec1, strVec2, sep = ", ")
```

```
strVec3
```

```
## [1] "Hello, Ma'am"      "Hi, Sir"          "What's up, Your Honor"
```

## • Numeric vectors

```
numVec1 <- c(30,50,70)
```

```
numVec1
```

```
## [1] 30 50 70
```

```
numVec2 <- seq(30,70,20)
```

```
numVec2
```

```
## [1] 30 50 70
```

```
numVec3 <- c(25,55,80)
```

```
numVec3
```

```
## [1] 25 55 80
```

```
numVec4 <- seq(from=20, to=1, by=-3)
```

```
numVec4
```

```
## [1] 20 17 14 11 8 5 2
```

```
2:6
```

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

## • min vs pmin?

```
min(numVec1) # by all
```

```
## [1] 30
```

```
min(numVec1, numVec3) # by all
```

```
## [1] 25
```

```
pmin(numVec1, numVec3) # by element
```

```
## [1] 25 50 70
```

```
numVec1 > numVec3 # by element
```

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

## • subsetting

```
numVec1[2]
```

```
## [1] 50
```

```
numVec1[-2]
```

```
## [1] 30 70
```

```
numVec1[1:2]
```

```
## [1] 30 50
```

```
numVec1[c(1,3)]
```

```
## [1] 30 70
```

## 2. array (matrix)

### 구조

- 동일한 data type의 벡터를 쌓아서 만든 데이터 구조
- (2D) 직각사각형의 데이터 구조
- (3D) 직육면체의 데이터 구조

### 생성

- matrix() 또는 array() 함수로 생성
  - data = c(9,2,3,4,5,6)로 element들을 나열
  - ncol = 3으로 3개의 컬럼을 가진 matrix 생성 (Number of COLUMN)
  - nrow로도 만들 수 있음 (Number of ROW)

```
mat <- matrix(data = c(9,2,3,4,5,6), ncol = 3)
mat
```

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

## Subsetting

- vector와 같은 방식으로 subsetting

```
mat[1, 2] # first row, second column
```

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

```
mat[2, ] # second row
```

```
## [1] 2 4 6
```

## 여러가지 방식으로 연산이 가능

- 원소단위, 행 단위, 열단위에 함수를 적용할 수 있다.
- `mean()`은 전체 element들에 대해서 평균을 구함
- `apply(MATRIX, 2, FUNCTION)`
  - MATRIX의 각 column에 FUNCTION을 apply
- `apply(MATRIX, 1, FUNCTION)`
  - MATRIX의 각 row에 FUNCTION을 apply

```
mat
##      [,1] [,2] [,3]
## [1,]    9    3    5
## [2,]    2    4    6
```

```
mean(mat)
```

```
## [1] 4.833333
```

```
apply(mat, 2, mean) # colMeans(mat)
```

```
## [1] 5.5 3.5 5.5
```

```
apply(mat, 1, mean) # rowMeans(mat)
```

```
## [1] 5.666667 4.000000
```

- `apply()` 함수는 왜 어렵게 느껴질까요?
  - 함수의 argument로 함수가 들어가서 어렵게 느껴짐
  - 과거에는 `apply()` 계열의 함수를 편리하게 사용할 수 있다면 중급사용자로 판단하기도 함.
  - `dplyr`, `tidyr` 등의 패키지의 함수들은 간편한 방식의 data wrangling 기능을 제공한다.




	Matrices		
	<pre>m &lt;- matrix(x, nrow = 3, ncol = 3)</pre> <p>Create a matrix from x.</p>		
<ul style="list-style-type: none"> <li>2번째 행</li> <li>1번째 열</li> <li>2번째 행의 3번째 원소</li> </ul>	 <pre>m[2, ]</pre> <p>- Select a row</p>	<pre>t(m)</pre> <p>Transpose:</p>	<ul style="list-style-type: none"> <li>행과 열을 바꿈</li> <li>행렬을 곱함</li> <li><math>Mx=n</math>의 방정식을 푸는 <math>x</math>를 찾음</li> </ul>
	 <pre>m[, 1]</pre> <p>- Select a column</p>	<pre>m %*% n</pre> <p>Matrix Multiplication:</p>	
	 <pre>m[2, 3]</pre> <p>- Select an element</p>	<pre>solve(m, n)</pre> <p>Find x in: <math>m * x = n</math>.</p>	

그림 5: Base Cheatsheet의 matrix관련 부분

### 3. data.frame

- vector를 모아서 네모나게 만든 것이 data.frame
  - data.frame() 함수를 이용
  - date, sky, temp, dust vector가 weather라는 data.frame의 column이 됨
  - data.frame을 생성할 때는 stringsAsFactors = FALSE 옵션을 넣어줌
  - (그렇지 않으면 string이 factor로 저장됨)

```
weather <-  
  data.frame(date = c("2017-8-31", "2017-9-1", "2017-9-2"),  
            sky  = c("Sunny", "Cloudy", "Rainy"),  
            temp = c(20, 15, 18),  
            dust = c(24, 50, 23),  
            stringsAsFactors = FALSE)  
  
weather
```

```
##      date    sky temp dust  
## 1 2017-8-31 Sunny  20   24  
## 2 2017-9-1 Cloudy  15   50  
## 3 2017-9-2 Rainy  18   23
```

- 엑셀과는 어떤 차이점이 있나요?
  - Column name과 Row name이 지정됨.
  - 즉, 데이터프레임 weather의 첫 번째 행, 첫 번째 열의 원소는 2017-08-31이다.



- 각 column은 변수에 해당하고 이름도 보존된다.
- `colnames()`로 `data.frame`의 각 column의 이름을 확인할 수 있다.
- `weather$sky`와 같이 특정 column을 이름을 사용해서 선택 가능
- `weather[,2]`와 같이 matrix의 subsetting 방법도 적용이 가능하다. 물론 `weather$sky`로 적는 것이 좋은 코딩 스타일이다.

```
colnames(weather)
```

```
## [1] "date" "sky" "temp" "dust"
```

```
weather$sky
```

```
## [1] "Sunny" "Cloudy" "Rainy"
```

```
weather$sky==weather[,2]
```

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

- 엑셀에서는 A열, B열, 2행, 3행 이렇게 column과 row를 정의했었지만...
- `data.frame`에서는 `colnames()`과 `rownames()`로 고유의 이름을 부여한다.

- `class()` 함수는 앞에서 다룬 data type 뿐 아니라 data structure를 확인하는 데에도 쓰인다.
- `class(VECTOR)`의 경우에는 vector 내의 각 element들의 data type을 확인할 수 있는데...
- `sapply()`를 data.frame에 적용하면 각 column에 같은 함수를 적용
- `sapply()`는 `apply` 계열 함수중에서 “simple apply”를 의미한다.

```
class(weather)
```

```
## [1] "data.frame"
```

```
class(weather$date)
```

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

```
sapply(weather, class)
```

```
##          date          sky          temp          dust
## "character" "character"  "numeric"  "numeric"
```

- date 벡터의 data type이 character 이므로 Date로 변환해보자.

```
weather$date <- as.Date(weather$date)
sapply(weather, class)
```

```
##          date          sky          temp          dust
## "Date" "character"  "numeric"  "numeric"
```

- `str()`은 데이터의 구조를 보여주는 함수이다.

```
str(weather)
```

```
## 'data.frame':   3 obs. of  4 variables:
## $ date: Date, format: "2017-08-31" "2017-09-01" ...
## $ sky : chr  "Sunny" "Cloudy" "Rainy"
## $ temp: num  20 15 18
## $ dust: num  24 50 23
```

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

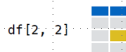
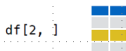
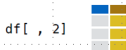


### Understanding a data frame

View(df) See the full data frame.

head(df) See the first 6 rows.

### Matrix subsetting



nrow(df)  
Number of rows.

ncol(df)  
Number of columns.

dim(df)  
Number of columns and rows.

**cbind** - Bind columns.



**rbind** - Bind rows.



- 데이터 프레임의 원소들은 길이가 같음
- 데이터 프레임은 *Matrix*와 *List*의 subsetting 명령어를 공유함

- Data.frame을 출력
- 처음 6개 행의 df를 보고 싶을때 (cf. tail(df))

- nrow, ncol, dim, head, tail, cbind, rbind는 대부분 프로그램에 아주 자주 사용하게 되는 명령어입니다.
- 이 명령어를 자주 사용하면 excel에서 R로 넘어온 두려움을 많이 없앨 수 있습니다.

그림 6: Base Cheatsheet의 data.frame관련 부분

## 4. list

```
HMSon <-  
  list(team = c("Korea", "Tottenham"),  
        birth = as.Date("1992-07-08"),  
        goals =  
          data.frame(team = c("U-17", "U-23", "A"),  
                      goals = c(4, 2, 7),  
                      stringsAsFactors = FALSE))  
  
HMSon  
  
## $team  
## [1] "Korea"      "Tottenham"  
##  
## $birth  
## [1] "1992-07-08"  
##  
## $goals  
##   team goals  
## 1 U-17     4  
## 2 U-23     2  
## 3  A       7
```

- list()로 다양한 데이터 구조를 함께 묶을 수 있다.
- 사용하기 어렵지만, 때로는 불가피하고 유용하다.

- `str()`로 tree형 구조를 파악할 수 있다.

```
str(HMSon)
```

```
## List of 3
## $ team : chr [1:2] "Korea" "Tottenham"
## $ birth: Date[1:1], format: "1992-07-08"
## $ goals:'data.frame': 3 obs. of 2 variables:
## ..$ team : chr [1:3] "U-17" "U-23" "A"
## ..$ goals: num [1:3] 4 2 7
```

- 이런 데이터 구조가 야근의 주범
- list형으로 데이터를 만들어야 한다면, 다시 한 번 잘 생각해본다. 정말로 불가피 한지...

- `names()`로 level-1 객체(바로 하위 레벨)의 이름 파악

```
names(HMSon)
```

```
## [1] "team" "birth" "goals"
```

- `sapply()`로 level-1 객체(바로 하위 레벨)에 동시에 함수 적용

```
sapply(HMSon, class)
```

```
##           team           birth           goals
## "character"    "Date" "data.frame"
```

- []로 subsetting 하면 여전히 list

```
HMSon[3]
```

```
## $goals
##   team goals
## 1 U-17      4
## 2 U-23      2
## 3   A       7
```

```
class(HMSon[3])
```

```
## [1] "list"
```

- [[]]로 subsetting 하면 level-1 객체를 반환

```
HMSon[[3]]
```

```
##   team goals
## 1 U-17      4
## 2 U-23      2
## 3   A       7
```

```
HMSon[["goals"]]
```

```
##   team goals
## 1 U-17      4
## 2 U-23      2
## 3   A       7
```

```
HMSon$goals
```

```
##   team goals
## 1 U-17      4
## 2 U-23      2
## 3   A       7
```

```
class(HMSon[[3]])
```

```
## [1] "data.frame"
```



## ● `sapply()`: “simple” apply

- 바로 하위 구조에 함수를 apply 한다.
- `data.frame` → 데이터프레임의 level-1 객체인 원소, 즉, 각각의 컬럼에 적용
- `list` → 리스트의 level-1 객체인 각 원소에 적용
- `vector` → 벡터의 level-1 객체인 각 원소에 적용

## 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]]`

Second element  
of l.

`l[1]`

New list with  
only the first  
element.

`l$x`

Element named  
x.

`l['y']`

New list with  
only element  
named y.

- `l`의 두번째 *element* (`y`가 벡터로 반환됨)
- `l`의 첫번째 *element* (`x`가 *list*로 반환됨)
- `l$x` 원소중 `x`라는 이름을 가진 것을 반환 (`x`가 *vector*로 반환됨)
- `l`중에서 이름이 `y`인 것을 반환 (`y`가 *list*로 반환됨)

- *List*의 *element*는 서로 다른 *type*과 길이일 수 있습니다.

그림 7: Base Cheatsheet의 list관련 부분

## 자료 구조 Summary

- Dimension: 점(0D), 선(1D), 면(2D)
- Homogeneous (동질성): level-1의 길이와 type이 같으면 동질적이다.
- Heterogenous (이질성): 동질적이지 않으면 이질적이다.

Dimension	Homogenous	Heterogenous
0D	element	N/A
1D	vector	list
$\geq 2D$	array	data.frame

- vector와 data.frame가 데이터 분석에서 흔히 자주 사용되는 구조이다.

## Control Statement

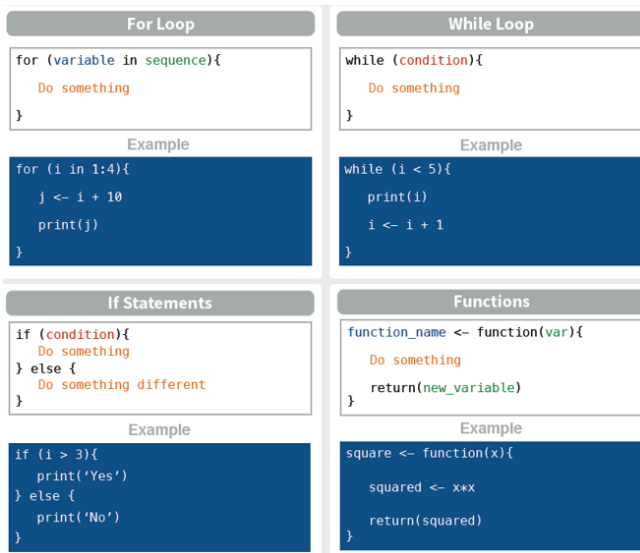


그림 8: Control Statements

# Base Cheatsheet

# Base R Cheat Sheet

## Getting Help

### Accessing the help files

#### 7mean

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.

## Using Packages

**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.

## Working Directory

**getwd()**

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

**setwd('C://file/path')**

Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

## Vectors

### Creating Vectors

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

### Vector Functions

<b>sort(x)</b>	<b>rev(x)</b>
Return x sorted.	Return x reversed.
<b>table(x)</b>	<b>unique(x)</b>
See counts of values.	See unique values.

### Selecting Vector Elements

#### By Position

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

#### By Value

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

#### Named Vectors

<b>x['apple']</b>	Element with name 'apple'.
-------------------	----------------------------

## Programming

### For Loop

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

#### Example

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

### While Loop

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

#### Example

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

## Reading and Writing Data

Also see the **readr** package.

Input	Output	Description
<b>df &lt;- read.table('file.txt')</b>	<b>write.table(df, 'file.txt')</b>	Read and write a delimited text file.
<b>df &lt;- read.csv('file.csv')</b>	<b>write.csv(df, 'file.csv')</b>	Read and write a comma separated value file. This is a special case of read.table/ write.table.
<b>load('file.Rdata')</b>	<b>save(df, file = 'file.Rdata')</b>	Read and write an R data file, a file type special for R.

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





## Suggested Readings

- R for Data Science
  - <https://r4ds.had.co.nz/>
- [R4DS] Ch2. Workflow: Basics
  - <https://r4ds.had.co.nz/workflow-basics.html>
- [R4DS] Ch4. Workflow: Scripts
  - <https://r4ds.had.co.nz/workflow-scripts.html>
- [R4DS] Ch11. Strings with `stringr`
  - <https://r4ds.had.co.nz/strings.html>
- [R4DS] Ch12. Factors with `forcats`
  - <https://r4ds.had.co.nz/factors.html>
- [R4DS] Ch15. Functions
  - <https://r4ds.had.co.nz/functions.html>
- [R4DS] Ch16. Vectors
  - <https://r4ds.had.co.nz/vectors.html>