

L02. R Basics

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



서울과학기술대학교 데이터사이언스학과

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

Section 1

Data Type

Motivation

주민등록번호

The diagram illustrates the structure of a Korean Resident Registration Number (주민등록번호). It is shown as '손흥민-920708-1234567'. Lines connect each part to a table below:

- '손흥민' connects to the '이름' (Name) column.
- '920708' connects to the '날짜' (Date) column.
- '1' connects to the '분류' (Category) column.
- '2345' connects to the '분류' (Category) column.
- '6' connects to the '숫자' (Number) column.
- '7' connects to the '숫자' (Number) column.

	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 주민등록번호에도 많은 정보가 담겨 있다.
- ❶ Data Type(문자, 날짜, 분류, 숫자)에 따라
- ❷ 필요한 처리의 종류가 다르며,
- ❸ 그에 맞는 함수를 사용해야 한다.

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 3 to 6

## [1] "says"
greeting # show

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

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

- 따옴표를 입력할때는 backslash를 앞에 붙여줍니다.
- **substr**은 SUBset of STRing, 즉, 문자열 변수의 부분 집합을 추출합니다.
- **cat**함수를 사용하면 backslash를 빼고 출력해줍니다.

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

Strings

Also see the **stringr** 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.

- **stringr** 패키지는 다른 여러 관련 함수가 있습니다.
- **String** 변수들을 합칩니다. (*vector* 포함)
- **String**으로 된 벡터의 원소 들을 합칩니다. (나중에)
- 문자열 *x*에 *pattern*이라는 문자열이 속해 있는가?
- 문자열 *x*의 *pattern*이라는 문자열을 *replace*로 교체
- 문자열 *x*의 모든 소문자를 대문자로 교체
- 문자열 *x*의 모든 문자의 개수를 세는 함수

Figure 1: String에 관련된 주요 함수

- 위의 함수들은 **base** 패키지의 함수들이고...
- **stringr**, **stringi**와 같은 패키지들이 더 빠른 속도로 **string**을 처리하는 함수를 제공한다.
- **stringr**, **stringi**는 **tidyverse** 패키지의 일부

2. numeric

- 그냥 계산기 처럼 사용할 수 있다.

```
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
```


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
```

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

Figure 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"
```

- 3줄의 코드에서 인수의 색이 다른 것이 보이시나요?
- 이 노트는 R문법을 이해하고 실행할 수 있는 `rmarkdown`을 이용해서 작성되었다.
- 강의 노트에서 문법이 틀린곳이 없다. (틀렸으면 컴파일이 안된다.)

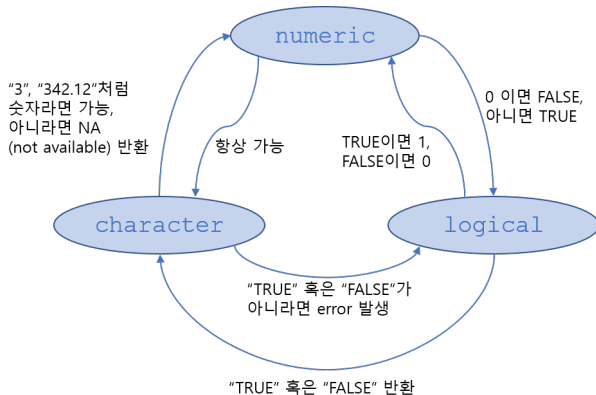


Figure 3: string-numeric-logical간의 변환

- 함수들 마다 제각각
 - ▶ 입력(input)에 해당하는 인수(argument)의 지정된 data type이 있다.
 - ▶ 출력(output)값도 지정된 data type으로 반환된다.
- 그렇기에 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.

Figure 4: Base Cheatsheet의 data type관련 부분

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

4. factor (Categorical, 범주형 변수)

● 데이터가 소속된 group을 나타내는 변수

- ▶ 숫자 vs Categorical
 - 더하고 뺄 수 있다면 **numeric**
 - 1,2,3을 A,B,C로 바꿔도 무리가 없다면 Categorical
- ▶ 문자 vs Categorical
 - Exclusive (배타적) 집합이고, 각 개체가 1개 그룹에 속한다면 Categorical
- ▶ Keywords
 - 'Partition', 'Classification', '분류', '집단', 'Group', '범주'
- ▶ 예시
 - 성인남자, 성인여자, 미성년자
 - 표준 산업 분류 (제조업, 금융업, 광공업등)
 - 날씨 (맑음, 흐림, 비가 옵)
- ▶ 생성
 - `data.frame` 생성시에 `stringsAsFactors = TRUE`를 사용하면 모든 문자 객체가 **factor**가 됨
 - 다른 data type에서 `as.factor()`를 이용해서 변환

Section 2

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? 뽕뽕뽕뽕한다?
- ② 몇 개의 관찰값이 있는가?
- ③ 어떤 규칙을 가지고 있는가?

Data Structure의 종류

- ❶ vector
- ❷ array (matrix)
- ❸ data.frame
- ❹ list

1. vector

- 길다랗게 저장되어 있는 데이터 구조
- `c()` 함수를 이용해서 벡터를 만들 수 있음
- `paste` 함수는 `string`으로 된 `vector`에도 적용이 가능!
- `seq()` 함수는 등차 수열을 만듦
- `a:b`는 `a`부터 `b`까지의 정수 벡터를 만듦

```
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"
```

```

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 (부분 선택)

```

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, 생성, subsetting

- (2D) 직각사각형의 데이터 구조
- (3D) 직육면체의 데이터 구조
- `matrix()` 또는 `array()` 함수로 생성
 - ▶ `data = c(9,2,3,4,5,6)`로 element들을 나열
 - ▶ `ncol = 3`으로 3개의 컬럼을 가진 matrix 생성 (Number of COLUMN)
 - ▶ `nrow`로도 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
```

- `vector`와 유사하게 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, 열단위)

- `mean()`은 전체 element들에 대해서 평균을 구함
- `apply(MATRIX, 2, FUNCTION)`
 - ▶ `MATRIX`의 각 column에 `FUNCTION`을 `apply` (적용)
- `apply(MATRIX, 1, FUNCTION)`
 - ▶ `MATRIX`의 각 row에 `FUNCTION`을 `apply` (적용)

```
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 판단한다라는 이야기도 있었으나...
 - ▶ `dplyr`, `tidyr` 등의 패키지의 함수를 이용하면 보다 간편한 방식으로 데이터 처리가 가능해짐




Matrices			
<pre>m <- matrix(x, nrow = 3, ncol = 3)</pre> <p>Create a matrix from x.</p>			
• 2번째 행	 <pre>m[2,]</pre> <p>- Select a row</p>	<pre>t(m)</pre> <p>Transpose:</p> <pre>m %*% n</pre> <p>Matrix Multiplication:</p> <pre>solve(m, n)</pre> <p>Find x in: $m \cdot x = n$</p>	• 행과 열을 바꿈
• 1번째 열	 <pre>m[, 1]</pre> <p>- Select a column</p>		• 행렬을 곱함
• 2번째 행의 3번째 원소	 <pre>m[2, 3]</pre> <p>- Select an element</p>		• $Mx=n$ 의 방정식을 푸는 x 를 찾음

Figure 5: Base Cheatsheet의 `matrix` 관련 부분

3. data.frame

- `vector`를 모아서 네모나게 만든 것이 `data.frame`
 - ▶ `data.frame()` 함수를 이용
 - ▶ `date`, `sky`, `temp`, `dust` `vector`가 `weather`라는 `data.frame`의 column이 됨
 - ▶ `data.frame`을 생성할 때는 `stringsAsFactors = FALSE` 옵션을 넣어줌
 - ▶ (그렇지 않으면 `string`이 `factor`로 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
```

- 엑셀화면이랑 비슷하죠? 어떤 차이점이 있나요?
 - ▶ Column name과 Row name이 지정된다!
 - ▶ 즉, `weather`의 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 코딩이다.

```
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...
- R의 `colnames()`과 `rownames()`는 2차원 데이터 구조에 이름을 부여한다.

- `class()` 함수는 앞에서 다룬 data type뿐 아니라 data structure를 확인하는 데에도 쓰인다.
- `class(VECTOR)`의 경우에는 vector내의 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 character이므로 Date로 변환해보자.

```
weather$date <- as.Date(weather$date)
```

- `str()` 함수는 데이터의 구조를 보여주므로 자주 사용

```
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

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.

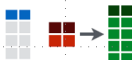
dim(df)

Number of columns and rows.

cbind - Bind columns.



rbind - Bind rows.



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

- Data.frame을 출력
- 처음 6개 행의 *df*를 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.

Figure 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 불가피한지...

- 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
- [][]로 subsetting 하면 level-1 객체를 반환

```
HMSon[3]
```

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

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

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

```
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
```

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

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

- `sapply()`: 'simple' apply

- ▶ 바로 하위 구조에 함수를 apply합니다.
- ▶ `data.frame` → 데이터프레임의 level-I 객체인 각 컬럼에 적용
- ▶ `list` → 리스트의 level-I 객체인 각 원소에 적용
- ▶ `vector` → 벡터의 level-I 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.

Figure 7: Base Cheatsheet의 list관련 부분

자료 구조 Summary

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

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

- **vector**와 **data.frame**가 데이터 분석에서 흔히 자주 사용되는 구조이다.

Section 3

Control Statement

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)
}
```

Section 4

Base Cheatsheet

Base R Cheat Sheet

Getting Help

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.

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

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)	rev(x)
Return x sorted.	Return x reversed.
table(x)	unique(x)
See counts of values.	See unique values.

Selecting Vector Elements

By Position

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'.
-------------------	----------------------------

Programming

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)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Output	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/write.table.
load('file.Rdata')	save(df, file = 'file.Rdata')	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

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

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.

Maths Functions

log(x)	Natural log.	sum(x)	Sum.
exp(x)	Exponential.	mean(x)	Mean.
max(x)	Largest element.	median(x)	Median.
min(x)	Smallest element.	quantile(x)	Percentage quantiles.
round(x, n)	Round to n decimal places.	rank(x)	Rank of elements.
signif(x, n)	Round to n significant figures.	var(x)	The variance.
cor(x, y)	Correlation.	sd(x)	The standard deviation.

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.

You can use the environment panel in RStudio to browse variables in your environment.

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	Now list with only the first element.	Element named x.	Now list with only element named y.


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

Matrix subsetting

 df[, 2]	 df[2, 1]	 df[2, 2]

List subsetting

 df\$x	 df[[2]]

Understanding a data frame

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

cbind - Bind columns.



rbind - Bind rows.



Strings

Also see the **stringr** package.

paste(x, y, sep = ' ')	Join multiple vectors together.
paste(x, collapse = ' ')	Join elements of a vector together.
grep(pattern, x)	Find regular expression matches in x.
gsub(pattern, replace, x)	Replace matches in x with a string.
toupper(x)	Convert to uppercase.
tolower(x)	Convert to lowercase.
nchar(x)	Number of characters in a string.

Factors

factor(x)	Turn a vector into a factor. Can set the levels of the factor and the order.
cut(x, breaks = 4)	Turn a numeric vector into a factor by 'cutting' into sections.

Statistics




lm(y ~ x, data=df)	Linear model.	t.test(x, y)	Test for a difference between means.	prop.test	Test for a difference between proportions.
glm(y ~ x, data=df)	Generalised linear model.	pairwise.t.test	Perform a t-test for paired data.	aov	Analysis of variance.
summary	Get more detailed information out a model.				

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poisson	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

Plotting

Also see the **ggplot2** package.

 plot(x)	Values of x in order.	 plot(x, y)	Values of x against y.	 hist(x)	Histogram of x.
---	-----------------------	--	------------------------	---	-----------------

Dates

See the **lubridate** package.

```
"Data Visualization"
```

```
## [1] "Data Visualization"
```