

## M12-base

LearningSpoonsR

2018-06-23

**M12-base (Hello World, Data Type, Data Structure, Control Statementss)**

## Contents

- Part 1. Hello World
- Part 2. 자료형 (Data Type)
- Part 3. 자료 구조 (Data Structure)
- Part 4. 프로그램 제어 (Control Statement)

# Part 1. Hello World

## Motivation

- 가장 간단한 프로그램을 작성하고 R문법을 탐색해 봅니다.
- 코딩은 GUI (Graphic User Interface)이 아니기 때문에 CPU와의 커뮤니케이션을 위해 문자로 된 메시지를 잘 다룰 수 있어야 합니다.
- 그렇기 때문에 “Hello World” 프로그램은 전통적으로 모든 프로그래머의 모든 프로그래밍 학습에 있어서 첫 번째 프로그램

## 문자 (String, Character)

- ✓ 1. 문자(String, Character)로 된 입력은 따옴표를 넣어줘야 합니다.

```
Hello World
## Error: unexpected symbol in "Hello World"
"Hello World"
## [1] "Hello World" ✓
```

- ✓ 2. 변수 입력 (Variable Assignment)

```
greeting <- "Hello World"
greeting
## [1] "Hello World"
```

3. 문자끼리는 덧셈 기호를 이용해서 더할수 없다.

```
greeting + "Sir"
## Error in greeting + "Sir" : non-numeric argument to binary operator
```

문자끼리 더하는 함수를 찾아야 한다!

## 변수 입력

```
a <- "apple"
```

- a라는 변수에 "apple"이라는 문자를 입력하는 명령 ✓
- a = "apple"을 사용해도 같음. 그러나 집어넣는다는 의미로서 "<-"를 권장
- 홑따옴표와 쌍따옴표의 기능은 대부분 경우에 같음

↗ astrgn "=="  
 ↗ 강의? "=="  
 a  
 ## [1] "apple"

- 현재 메모리에 a라는 변수에 어떤 값이 들어있는지 확인
- print(a)도 같은 기능을 함.
- cat(a)도 거의 같은 기능을 함. (문장 부호가 많이 쓰인 경우에 좀더 단정하게 출력)

## 변수의 이름

- 변수의 이름은 영어로 하는 것이 바람직함.
- 특수 문자는 대부분 사용이 불가능 함. (예외: "\_", ".")
- 너무 길지 않으면서 의미가 잘 전달되게 지어야 함.
- 이름 짓는 방식

1. learningspoons: 가독성이 나쁨
2. learningSpoons: Camel 방식이라고 함. 가독성이 좋아 추천. R에서 가장 추천되는 방식
3. learning\_spoons: 전통적인 방식으로 대소문자 구분이 없던 시절부터 사용됨. 아직도 많이 사용되고 있음.

“좋은 프로그래머가 되려면 두 가지를 잘해야 하는데, 하나는 메모리를 잘 관리하는 것이고, 다른 하나는 이름을 잘 짓는 것이다”. - from Advanced R

## 함수 (Function)

함수는

1. 입력 (Input)을 가지고
2. 어떤 행동을 수행하고
3. 그 결과로서 출력(Output)을 반환(Return)하는 것

대표적인 함수의 기능

1. 처리 (process)
2. 생성 (generate, populate)
3. 변환 (convert)
4. 표시 (display, print)
5. 합치기 (aggregate, combine, concatenate)
6. 추출 (filter)
7. 저장 (save, write)
8. 불러오기 (load, infile)

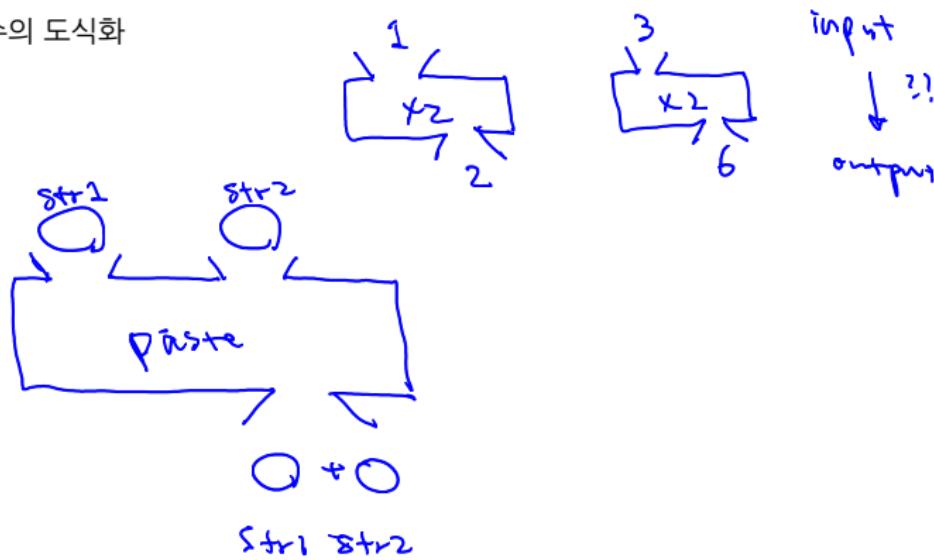
merge

## 문자 두 개를 합쳐주는 함수?

- google “R function combine two strings”
- google “R에서 문자를 합치는 함수”

R 문자 합치기

- 여러분은 이제 모든 프로그래밍 언어에서 문자를 합하고 출력하는 방법을 알고 있습니다?!
- 함수의 도식화



## 변수(variable)와 함수(function)

1. 변수

greeting

- 존재하는 것
- 이름은 명사로 지어야 함

변수



2. 함수

paste

- 행동하는 것
- 이름은 동사로 지어야 함

함수



3. 데이터 사이언스의 대부분 프로그램은 변수로 시작해서 함수가 계속 작동하면서 최종적으로 결론에 해당하는 변수를 만들어 내는 것

## paste 함수

```
paste(greeting, "Sir")
## [1] "Hello World Sir"
paste(greeting, "!")
## [1] "Hello World !"
paste0(greeting, "!")
## [1] "Hello World!"
paste(greeting, "!", sep = "")
## [1] "Hello World!"
```

1. 더하기로는 문자를 합칠 수 없습니다. (파이썬에서는 됩니다.)
2. “+”는 binary operator라고 합니다.
3. paste 함수는 문자열을 합칩니다.
4. paste0을 사용하면 띄어쓰기 없이 합쳐집니다.
5. sep 옵션을 사용하면 paste함수로도 띄어쓰기를 안 할 수 있습니다.

blank

## Part 2. 자료형 (Data Type)

문자

숫자.

# 1. String (Character, 문자열)

```

greeting <- 'R says \Hello World!\'
nchar(greeting)
## [1] 21
substr(greeting, 3, 6)
## [1] "says"
greeting
## [1] "R says \Hello World!\"
cat(greeting)
## R says "Hello World!"

```

number of character  
 subset of string

- 따옴표를 입력할때는 “”(backslash)를 앞에 붙여줍니다.
- `substr`은 SUBset of STRing, 즉, 문자열 변수의 부분 집합을 추출합니다.
- `cat`함수를 사용하면 “”(backslash)를 빼고 출력해줍니다.

```

paste(string1, string2)
paste0(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.

- **stringr** 패키지는 다른 여러 관련 함수가 있습니다.
- **String** 변수들을 합칩니다. (**vector** 포함)
- **String**으로 된 벡터의 원소들을 합칩니다. (나중에)
- 문자열 x에 pattern이라는 문자열이 속해 있는가?
- 문자열 x의 pattern이라는 문자열을 replace로 교체
- 문자열 x의 모든 소문자를 대문자로 교체
- 문자열 x의 모든 문자의 개수를 세는 함수

Figure 1: String에 관련된 주요 함수

- Cheatsheet을 가까이 하세요 (ex. 책상이나 파티션에 붙여둠)
- 설명서가 읽기 귀찮으면 Trial & Error도 좋은 방법입니다.

## 2. numeric

$10^2 + 36$

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

a = 4

a

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

a\*5

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

a = a + 10

a

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

$$\textcircled{a=4}$$

$$\textcircled{a \leftarrow a + 10}$$

### 3. logical

`2==3`

`## [1] FALSE`

`5>3`

`## [1] TRUE`

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

Figure 2: logical 값을 반환하는 수치 비교 함수

## Data Type 확인과 변환

```

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
    
```



## 4. factor (Categorical, 범주형 변수)

### 1. Group 별로 서로 예외적인(exclusive) 특성을 표현한 것

- 남자, 여자, 미성년 (여기에서 남자와 여자는 성인이어야 함)
- 산업 분류: 제조업, 금융업, 광공업 등
- 맑음, 흐림, 비가 옴, 더움
- 숫자로 되어있을 경우에 A,B,C로 바꿔도 무리가 없다면 Categorical 변수

### 2. String vs Factor

- Name/Message(이름/단문) vs Classification/Category(분류/속성)
- 일반적으로 유일한 값 vs 몇 가지 비슷한 것이 존재

## 5. Date

```
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
mydates[1] - mydates[2]

## Time difference of 1225 days
today <- Sys.Date( )
today

## [1] "2018-06-23"
as.numeric(substr(today, 1, 4)) # year # substr works for date

## [1] 2018
as.numeric(substr(today, 6, 7)) # month

## [1] 6
as.numeric(substr(today, 9, 10)) # day

## [1] 23
format(today, format="%B %d %Y")

## [1] "6 23 2018"
```

## Data Type 변환

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.

- 아래 표에서 위에서 아래의 방향으로만 변환이 가능합니다.
- TRUE와 FALSE 값 (이를 Boolean Value라고 함)
- 정수(Integer) 혹은 소수(Floating Point Numbers)
- 문자열. 일반적으로 factor 보다 선호됨
- 문자인데 지정된 값만을 가지는 범주형 데이터. 몇몇 통계 모형에 필요함.

Figure 3: Base Cheatsheet의 type관련 부분

- 데이터 타입의 개수도 매우 많습니다.
- 새로운 패키지마다 패키지에 적합한 타입을 나름대로 정의하기도 합니다.
- 마주칠때마다 검색하고 필요한 만큼 알아보고 사용하는 것이 프로그래밍입니다.

## 예시 - 주민등록번호

YYMMDD - 1ABCD1E

- YYMMDD - 생년 월일
- 1 - 출생 연대와 성별
- ABCD - 출생등록지 고유번호
- AB - 광역시도 고유번호
- CD - 읍면동 고유번호
- 1 - 일련번호
- E - 검증번호
- $E = 11 - \{(2 \times ㄱ + 3 \times ㄴ + 4 \times ㄷ + 5 \times ㄹ + 6 \times ㅁ + 7 \times ㅂ + 8 \times ㅅ + 9 \times ㅇ + 2 \times ㅈ + 3 \times ㅊ + 4 \times ㅋ + 5 \times ㅌ\} \bmod 11\}$

blank

## Part 3. 자료 구조 (Data Structure)

## 자료형 (type) vs 자료구조 (structure)

### 1. 자료형

- 변수에 입력된 하나의 값의 형태
- 즉, 0차원의 하나의 점
- 하나의 값, singleton
- 그런데 하나의 값마다 하나의 변수가 되어 이름을 가진다면?
- 엑셀도 A컬럼 1번행 등으로 묶어서 처리하는 기능을 제공

### 2. 자료 구조

- 각각의 값(singleton)들이 모여서 구성되어 있는 구조를 자료 구조라고 함.
- 대용량 데이터도 한 번에 포함할 수 있기에 데이터 분석의 근간이 됨.

# 1. Vector

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

##      strVec1
## [1,] "Hello"
## [2,] "Hi"
## [3,] "What's up"
strVec2 <- c("Ma'am", "Sir", "Your Honor")
cbind(strVec2)

##      strVec2
## [1,] "Ma'am"
## [2,] "Sir"
## [3,] "Your Honor"
strVec3 <- paste(strVec1, strVec2)
cbind(strVec3)

##      strVec3
## [1,] "Hello Ma'am"
## [2,] "Hi Sir"
## [3,] "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(numVec1)
```

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

```
min(numVec1,numVec3)
```

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

```
pmin(numVec1,numVec3)
```

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

pmin(numVec1 > numVec3)

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

```
numVec1[2]
```

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

```
numVec1[-2]
```

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

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

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

numVec1[c(1,3)]

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

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

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

- 원소를 나열하여 벡터 생성
- 1간격으로 증가하는 정수로 벡터 생성
- 등차수열의 벡터 생성
- 벡터를 반복
- 벡터의 원소들을 반복
- 정렬 (낮음차순/오름차순); 거꾸로 뒤집기
- 빈도를 표로 표현; 중복되는 값을 제거하여 반환

Figure 4: Base Cheatsheet의 vector관련 부분(1)

Selecting Vector Elements	
	By Position
<code>x[4]</code>	The fourth element.
<code>x[-4]</code>	All but the fourth.
<code>x[2:4]</code>	Elements two to four.
<code>x[-(2:4)]</code>	All elements except two to four.
<code>x[c(1, 5)]</code>	Elements one and five.
	By Value
<code>x[x == 10]</code>	Elements which are equal to 10.
<code>x[x &lt; 0]</code>	All elements less than zero.
<code>x[x %in% c(1, 2, 5)]</code>	Elements in the set 1, 2, 5.
	Named Vectors
<code>x['apple']</code>	Element with name 'apple'.
<ul style="list-style-type: none"> <li>4번째 원소만</li> <li>4번째 원소만 빼고</li> <li>2번째부터 4번째 원소까지</li> <li>2번째부터 4번째 원소들을 빼고</li> <li>1번째와 5번째 원소만</li> <li>값이 10인 원소만 선택</li> <li>0보다 작은 원소만 선택</li> <li>x가 1,2,5중에 하나인 경우만 선택</li> <li>string의 경우에는 따옴표를 사용할 수 있음</li> </ul>	

$x \leftarrow c(3, 6, 5, 10)$   
 $x == 10$  [F T F T]  
 $x[x == 10]$

`which(x == 10)`

Figure 5: Base Cheatsheet의 vector 관련 부분(2)

Maths Functions					
• 자연로그	<code>log(x)</code>	Natural log.	<code>sum(x)</code>	Sum.	
• 지수함수	<code>exp(x)</code>	Exponential.	<code>mean(x)</code>	Mean.	• 중간값
• 최대/최소값	<code>max(x)</code>	Largest element.	<code>median(x)</code>	Median.	• 백분위수
• 소수점 n자리까지 반올림	<code>min(x)</code>	Smallest element.	<code>quantile(x)</code>	Percentage quantiles.	• 순위
• n개의 유효숫자	<code>round(x, n)</code>	Round to n decimal places.	<code>rank(x)</code>	Rank of elements.	• 분산
• 상관관계	<code>signif(x, n)</code>	Round to n significant figures.	<code>var(x)</code>	The variance.	• 표준편차
	<code>cor(x, y)</code>	Correlation.	<code>sd(x)</code>	The standard deviation.	

Figure 6: Base Cheatsheet의 numeric vector 관련 부분

## 2. matrix (array)

```
mat=matrix(data=c(9,2,3,4,5,6),ncol=3)
mat
##      [,1] [,2] [,3]
## [1,]     9     3     5
## [2,]     2     4     6
mat[1,2]
## [1] 3
mat[2,]
## [1] 2 4 6
mean(mat)
## [1] 4.833333
apply(mat, 2, mean)      mat의 column wise mean ???? , apply
## [1] 5.5 3.5 5.5
apply(mat, 1, mean)
## [1] 5.666667 4.000000
```

**Matrices**

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

- 2번째 행
- 1번째 열
- 2번째 행의 3번째 원소

	<code>m[2, ]</code> - Select a row	<code>t(m)</code> Transpose	• 행과 열을 바꿈
	<code>m[, 1]</code> - Select a column	<code>m %*% n</code> Matrix Multiplication	• 행렬을 곱함
	<code>m[2, 3]</code> - Select an element	<code>solve(m, n)</code> Find x in: $m \cdot x = n$	• $Mx=n$ 의 방정식을 푸는 x를 찾음

Figure 7: Base Cheatsheet의 matrix관련 부분

### 3. data.frame

```
weather <-  
  data.frame(date = c("2017-8-31", "2017-9-1", "2017-9-2"),  
             sky = c("Sunny", "Cloudy", "Rainy"),  
             temp = c(20, 15, 18))  
  
weather  
##      date    sky  temp  
## 1 2017-8-31  Sunny   20  
## 2 2017-9-1 Cloudy   15  
## 3 2017-9-2 Rainy   18  
  
colnames(weather)  
## [1] "date" "sky"  "temp"  
weather$sky  
## [1] Sunny  Cloudy Rainy  
## Levels: Cloudy Rainy Sunny  
weather$sky==weather[,2]  
## [1] TRUE TRUE TRUE
```

```
class(weather$date)
## [1] "factor"
weather$date <- as.Date(weather$date)
weather$sky <- as.character(weather$sky)
weather

##          date    sky   temp
## 1 2017-08-31  Sunny    20
## 2 2017-09-01 Cloudy    15
## 3 2017-09-02 Rainy    18
sapply(weather, class)

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

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

<code>df[, 2]</code>	
<code>df[2, ]</code>	
<code>df[2, 2]</code>	

### List subsetting

<code>df\$x</code>	
<code>df[[2]]</code>	

### Understanding a data frame

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

### cbind - Bind columns.



### rbind - Bind rows.



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

- `Data.frame`을 출력

- 처음 6개 행의 `df`를 보고 싶을때  
(cf. `tail(df)`)

- `nrow`, `ncol`, `dim`, `head`, `tail`, `cbind`, `rbind`는 대 부분 프로그램에 아주 자주 사용하게 되는 명령어입니다.

- 이 명령어를 자주 사용하면 excel에서 R로 넘어온 두려움을 많이 없앨 수 있습니다.

Figure 8: Base Cheatsheet의 `data.frame`관련 부분

- `data.frame`을 생성할 때(데이터 파일을 불러올 때)에는 문자로된 데이터를 자동으로 `factor`로 처리함!
- 따라서 `stringsAsFactors=FALSE`의 option을 `data.frame` 생성시 추가하는 것을 권장
- 혹은 `options(stringsAsFactors=FALSE)` 명령을 global option으로 사용 가능
- `factor` 변수로 원할때에는 `as.factor()`를 사용하여 변환
- Date인데 string으로 분류된 변수들에 대해서 `as.Date()`를 이용하여 type변환

## 5. list

```
sim <- list(numClubs = c(1, 8, 3, 1),
            car      = c("Mercedez", "Porche", "Kia"),
            hobby    = c("golf", "squash"))
sim
## $numClubs
## [1] 1 8 3 1
##
## $car
## [1] "Mercedez" "Porche"   "Kia"
##
## $hobby
## [1] "golf"     "squash"
names(sim)
## [1] "numClubs" "car"       "hobby"
sim[[1]]
## [1] 1 8 3 1
sim$numClubs * c(10, 5, 7, 20)
## [1] 10 40 21 20
```

**Lists**

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

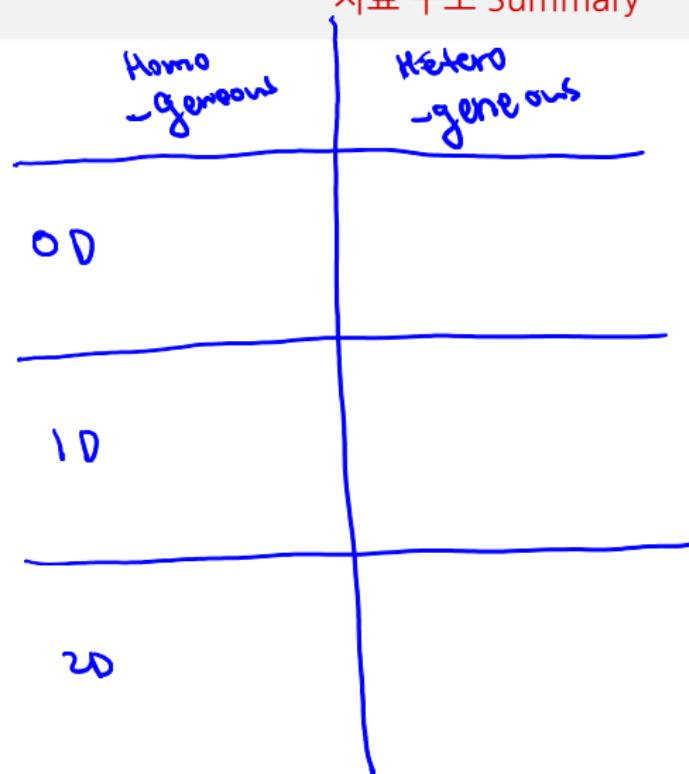
<code>l[[2]]</code>	<code>l[1]</code>	<code>l\$x</code>	<code>l['y']</code>
Second element of <code>l</code> .	New list with only the first element.	Element named <code>x</code> .	New list with only element named <code>y</code> .

- `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 9: Base Cheatsheet의 list관련 부분

## 자료 구조 Summary



blank

## Part 4 프로그램 제어 (Control Statement)

For Loop	While Loop
<pre>for (variable in sequence){     Do something }</pre>	<pre>while (condition){     Do something }</pre>
Example	Example
<pre>for (i in 1:4){     j &lt;- i + 10     print(j) }</pre>	<pre>while (i &lt; 5){     print(i)     i &lt;- i + 1 }</pre>
If Statements	Functions
<pre>if (condition){     Do something } else {     Do something different }</pre>	<pre>function_name &lt;- function(var){     Do something     return(new_variable) }</pre>
Example	Example
<pre>if (i &gt; 3){     print('Yes') } else {     print('No') }</pre>	<pre>square &lt;- function(x){      squared &lt;- x*x      return(squared) }</pre>

Figure 10: Control Statements

- `paste0` 함수, `print` 함수, `for`문을 활용하여 트리를 그리는 함수를 작성해 보세요.
- `paste0`함수의 `collapse` 옵션을 사용하세요.

```
function <- tree(h) {  
  ...  
  ...  
  ...  
  ...  
  ...  
  ...  
}
```

```
tree(4)
```

```
## [1] "     A     "  
## [1] "     AAA    "  
## [1] "     AAAAA   "  
## [1] "     AAAAAAA  "
```

```
tree(6)
```

```
## [1] "         A         "  
## [1] "         AAA       "  
## [1] "         AAAAA      "  
## [1] "         AAAAAAA    "  
## [1] "         AAAAAAAA   "  
## [1] "         AAAAAAAAA  "
```

Figure 11: For문 실습

blank

# Cheatsheet for Base

## 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		Programming	
Creating Vectors		For Loop	
<code>c(2, 4, 6)</code>	2 4 6 Join elements into a vector	<code>for (variable in sequence){   Do something }</code>	<code>while (condition){   Do something }</code>
<code>2:6</code>	2 3 4 5 6 An integer sequence	<b>Example</b>	<b>Example</b>
<code>seq(2, 3, by=0.5)</code>	2.0 2.5 3.0 A complex sequence	<code>for (i in 1:4){   j &lt;- i + 10   print(j) }</code>	<code>while (i &lt; 5){   print(i)   i &lt;- i + 1 }</code>
<code>rep(1:2, times=3)</code>	1 2 1 2 1 2 Repeat a vector		
<code>rep(1:2, each=3)</code>	1 1 1 2 2 2 Repeat elements of a vector		
Vector Functions		While Loop	
<code>sort(x)</code>	<code>rev(x)</code>	<code>if (variable == value){   Do something }</code>	<code>if (condition){   Do something }  <code>else {   Do something different }</code></code>
Return x sorted.	Return x reversed.	<b>Example</b>	<b>Example</b>
<code>table(x)</code>	<code>unique(x)</code>	<code>if (i &gt; 3){   print('Yes') } else {   print('No') }</code>	<code>function_name &lt;- function(var){   Do something   return(new_variable) }</code>
See counts of values.	See unique values.		
Selecting Vector Elements		Functions	
By Position		If Statements	
<code>x[4]</code>	The fourth element.	<code>if (variable == value){   Do something }</code>	<code>if (variable == value){   Do something }</code>
<code>x[-4]</code>	All but the fourth.	<b>Example</b>	<b>Example</b>
<code>x[2:4]</code>	Elements two to four.	<code>if (i &gt; 3){   print('Yes') } else {   print('No') }</code>	<code>square &lt;- function(x){   squared &lt;- x*x   return(squared) }</code>
<code>x[-(2:4)]</code>	All elements except two to four.		
<code>x[c(1, 5)]</code>	Elements one and five.		
By Value		Reading and Writing Data	
<code>x[x == 10]</code>	Elements which are equal to 10.	<b>Input</b>	<b>Output</b>
<code>x[x &lt; 0]</code>	All elements less than zero.	<code>df &lt;- read.table('file.txt')</code>	<code>write.table(df, 'file.txt')</code>
<code>x[x %in% c(1, 2, 5)]</code>	Elements in the set 1, 2, 5.		Read and write a delimited text file.
Named Vectors		<b>Description</b>	
<code>x['apple']</code>	Element with name apple.	<code>df &lt;- read.csv('file.csv')</code>	<code>write.csv(df, 'file.csv')</code>
			Read and write a comma separated value file. This is a special case of read.table/write.table.
		<code>load('file.RData')</code>	<code>save(df, file = 'file.Rdata')</code>
			Read and write an R data file, a file type special for R.
Conditions		Also see the <a href="#">readr package</a> .	
<code>a == b</code>	Are equal	<code>a &gt; b</code>	Greater than
<code>a != b</code>	Not equal	<code>a &lt; b</code>	Less than
<code>a &lt;= b</code>		<code>a &lt;= b</code>	Greater than or equal to
<code>a &gt;= b</code>		<code>a &gt;= b</code>	Less than or equal to
<code>is...na(x)</code>		<code>is...null(x)</code>	Is missing
		<code>is...null(x)</code>	Is null

RStudio® is a trademark of RStudio, Inc. • [CC BY](#) Mhammadiwell • mhammadiwell@gmail.com

Learn more at [web page](#) or [vignette](#) • package version • Updated: 3/13

47 / 49

### 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.

### Matrices

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

	<code>m[2, ]</code> - Select a row	<code>t(m)</code> - Transpose
	<code>m[, 1]</code> - Select a column	<code>m %*% n</code> - Matrix Multiplication
	<code>m[2, 3]</code> - Select an element	<code>solve(m, x)</code> - Find x in: $m \cdot x = n$

### Strings

Also see the [string](#) 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.

### Maths Functions

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

### Lists

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

<code>l[[1]]</code>	<code>l[1]</code>	<code>l\$x</code>	<code>l["y"]</code>
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the [dplyr](#) package.

### Factors

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

### 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

<code>df[, 2]</code>	<code>df\$x</code>	<code>df[, 2]</code>
df[ , 2]	View(df)	head(df)
df[2, ]	See the full data frame.	See the first 6 rows.
df[2, ]	<code>nrow(df)</code> Number of rows.	<code>cbind</code> - Bind columns.
df[2, ]	<code>ncol(df)</code> Number of columns.	<code>rbind</code> - Bind rows.
df[, 2]	<code>dim(df)</code> Number of columns and rows.	

Understanding a data frame

### Statistics

<code>lm(y ~ x, data=df)</code>	Linear model.
<code>glm(y ~ x, data=df)</code>	Generalised linear model.
<code>summary</code>	Get more detailed information out a model.
<code>pairwise.t.test</code>	Perform a t-test for paired data.
<code>aoov</code>	Analysis of variance.

### Distributions

	Random Variables	Density Function	Cumulative Distribution	Quantile
Normal	<code>rnorm</code>	<code>dnorm</code>	<code>pnorm</code>	<code>qnorm</code>
Poisson	<code>rpois</code>	<code>dpois</code>	<code>ppois</code>	<code>qpois</code>
Binomial	<code>rbinom</code>	<code>dbinom</code>	<code>pbinom</code>	<code>qbinom</code>
Uniform	<code>runif</code>	<code>dunif</code>	<code>runif</code>	<code>qunif</code>

### Plotting

Also see the [ggplot2](#) package.

<code>plot(x)</code>	Values of x in order.
<code>plot(x, y)</code>	Values of x against y.
<code>hist(x)</code>	Histogram of x.

### Dates

See the [lubridate](#) package.

Learn more at [web page](#) or [vignette](#) • package version • updated: 3/10

```
> 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.

M12-base (Hello World, Data Type, Data Structure,  
Control Statements)