



Lecture 1

Hello World *Base 문법*

러닝스폰즈

2018년 4월

가장 간단한 프로그램을 작성하고
 R 문법을 천천히 탐색해 봅시다.

The slide features a light gray background with a subtle gradient. In the top-left and bottom-right corners, there are several realistic water droplets of varying sizes, rendered with soft shadows and highlights to give them a three-dimensional appearance. The text "Module 1: Hello World" is centered in the middle of the slide.

Module 1: Hello World

Hello World?

- 코딩은 *GUI* 환경이 아니기 때문에 *CPU*와의 커뮤니케이션을 위해 문자로 된 메시지를 잘 다룰 수 있어야 함.
- 그렇기 때문에 “*Hello World*” 프로그램은 전통적으로 모든 프로그래머의 모든 프로그래밍 학습에 있어서 첫 번째 프로그램

Hello World

```
Hello World ↵  
## Error: unexpected symbol in "Hello World" ↵
```

```
"Hello World" ↵  
## [1] "Hello World" ↵
```

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

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

1. 문자(*String, Character*)로 된 입력은 따옴표를 넣어줘야 합니다.
2. 변수 입력
(*Variable Assignment*)
3. 문자끼리는 덧셈기호를 이용해서 더할 수 없다.
4. 문자끼리 더하는 함수를 찾아야 한다.
(*Function*)

변수 입력 (*Variable Assignment*)

Variable Assignment

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

- $a <- 'apple'$
 - : a 라는 변수에 “apple”이라는 문자를 입력하는 명령
 - : $a='apple'$ 을 사용해도 같음. 그러나 R에서는 $<-$ 를 권장
 - : 홀따옴표와 쌍따옴표의 기능은 대부분 같음
- a
 - : 현재 메모리에 a 라는 변수가 잘 입력되어 있는지 확인

변수의 이름

- 변수의 이름은 영어로 하는 것이 바람직함.
- 특수문자 사용 불가
- 이름을 짓는 방식
 - *learningspoons* : 가독성이 나쁨
 - *learningSpoons* : Camel 방식. 가장 가독성 좋음. R에서 가장 추천됨.
 - *learning_spoons* : 전통적인 방식. 대/소문자 구분이 불가능 하던 시대에 부터 많이 사용됨. 아직도 많은 프로그래머들이 사용하는 방식

함수 (*Function*)

- 함수(*function*)는
 - 입력(*Input*)을 가지고
 - 어떤 행동을 수행하고
 - 그 결과로서 출력(*Output*)을 반환하는 것
- 대표적인 함수의 기능
 - 처리(*processing*)
 - 생성(*generating*)
 - 변환(*converting*)
 - 표시(*display*)
 - 합치기(*aggregate, combine*)
 - 추출(*filter*)

문자 두 개를 합쳐주는 함수?

- 구글 검색 생활화!
 - Google *"R function combine two string"*
 - Google *"R에서 문자를 합치는 함수"*
- 여러분은 이제 모든 프로그래밍 언어에서 *"Hello World"*를 출력하는 방법을 알고 있습니까?!
- 함수의 도식화

변수와 함수

- 변수
 - 존재하는 것
 - 이름은 명사로
- 함수
 - 행동하는 것
 - 이름은 동사로
- *“Something that exists is a variable, something that does something is a function”.*
- 데이터 사이언스의 대부분 프로그램은 변수를 가지고 함수가 계속 작동하면서 최종적으로 결론에 해당하는 변수를 만들어 내는 것

Paste 함수 사용 예시

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

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 operator*라고 합니다.
3. *paste* 함수는 문자열을 합칩니다.
4. *paste0*을 사용하면 띄어쓰기 없이 합쳐집니다.
5. *sep* 옵션을 사용하면 *paste* 함수로도 띄어쓰기를 안 할 수 있습니다.

Module 2: 자료형 (Data Type)

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!"
```

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

- 따옴표를 입력할 때는 `\`(backslash)를 앞에 붙여줍니다.
- *substr*은 *SUB*set of *STR*ing, 즉, 문자열 변수의 부분 집합을 추출합니다.
- *cat*함수를 사용하면 `\`(backslash)를 빼고 출력해줍니다.

1. String (Character, 문자열)

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 *replac*로 교체
- 문자열 *x*의 모든 소문자를 대문자로 교체
- 문자열 *x*의 모든 문자의 개수를 세는 함수

- 설명서가 읽기 귀찮으면 *Trial & Error*도 좋은 방법입니다.
- 수업중의 *CheatSheet* 부분에 대해서 번역을 제공하니 프로그래밍을 위한 영어에 익숙해지셨으면 합니다.

2. Numeric (숫자)

```
10^2 + 36 ↵  
## [1] 136 ↵  
  
a <- 4 ↵  
a ↵  
  
## [1] 4 ↵  
  
a*5 ↵  
  
## [1] 20 ↵  
  
a <- a + 10 ↵  
a ↵  
  
## [1] 14 ↵
```

a 변수에 a+10이라는 값을 넣어줌

3. Logical (boolean)

```
2==3 ↵ 2와 3이 같으면 참, 다르면 거짓  
## [1] FALSE ↵  
  
5>3 ↵  
## [1] TRUE ↵
```

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

Data Type 변환 (Conversion)

```
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 자료)

- *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 올, 더움
 - 숫자로 되어있을 경우에 A,B,C로 바뀌도 무리가 없다면 *Categorical Data*
- *String vs Factor*
 - *Name/Message*(이름/단문) vs *Classification/Category*(분류/속성)
 - 일반적으로 유일한 값 vs 몇 가지 비슷한 것이 존재

5. Date (날짜와 시간)

```
mydates <- as.Date(c("2007-06-22", "2004-02-13"))  
mydates[1] - mydates[2]  
  
## Time difference of 1225 days  
  
today <- Sys.Date()  
today  
  
## [1] "2018-04-15"  
  
as.numeric(substr(today,1,4)) # substr works for date  
  
## [1] 2018  
  
as.numeric(substr(today,6,7)) # month  
  
## [1] 4  
  
as.numeric(substr(today,9,10)) # day  
  
## [1] 15  
  
format(today, format="%B %d %Y")  
  
## [1] "4 월 15 2018"  
  
## SEE lubridate package
```

Data Type 변환 (Conversion)

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.
- *TRUE*와 *FALSE* 값 (이를 *Boolean Value*라고 함)
- 정수(*Integer*) 혹은 소수(*Floating Point Numbers*)
- 문자열. 일반적으로 *factor* 보다 선호됨
- 문자인데 지정된 값만을 가지는 범주형 데이터. 몇몇 통계 모형에 필요함.

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

Data Type - 주민등록번호

- *YYMMDD - 1ABCD1E*
 - *YYMMDD* - 생년 월일
 - *1* - 출생 연대와 성별
 - *ABCD* - 출생등록지 고유번호
 - *AB* - 광역시도 고유번호
 - *CD* - 읍면동 고유번호
 - *1* - 일련번호
 - *E* - 검증번호
 - $E = 11 - \{(2 \times \text{ㄱ} + 3 \times \text{ㄴ} + 4 \times \text{ㄷ} + 5 \times \text{ㄹ} + 6 \times \text{ㅁ} + 7 \times \text{ㅂ} + 8 \times \text{ㅅ} + 9 \times \text{ㅇ} + 2 \times \text{ㅈ} + 3 \times \text{ㅊ} + 4 \times \text{ㅋ} + 5 \times \text{ㅌ}) \bmod 11\}$
- *MOD?*

Module 3: 자료 구조(Data Structure)

자료형 vs 자료구조

- 자료형

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

- 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"↵
```

1. Vector

```
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 ↓
```


1. Vector

Vectors

Creating Vectors

<code>c(2, 4, 6)</code>	2 4 6	Join elements into a vector
<code>2:6</code>	2 3 4 5 6	An integer sequence
<code>seq(2, 3, by=0.5)</code>	2.0 2.5 3.0	A complex sequence
<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

`sort(x)`

Return x sorted.

`table(x)`

See counts of values.

`rev(x)`

Return x reversed.

`unique(x)`

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

1. Vector

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'.

- 4번째 원소만
- 4번째 원소만 빼고
- 2번째 부터 4번째 원소까지
- 2번째 부터 4번째 원소들을 빼고
- 1번째와 5번째 원소만
- 값이 10인 원소만 선택
- 0보다 작은 원소만 선택
- x 가 1,2,5중에 하나인 경우만 선택
- *string*의 경우에는 따옴표를 사용할 수 있음

1. Vector

- 자연로그
- 지수함수
- 최대/최소값
- 소수점 n 자리까지 반올림
- n 개의 유효숫자
- 상관관계

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.

- 중간값
- 백분위수
- 순위
- 분산
- 표준편차

2. Matrix

```
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 ↵
```




2. Matrix

- 2번째 행
- 1번째 열
- 2번째 행의 3번째 원소

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, n)</code> Find x in: $m * x = n$

- 행과 열을 바꿈
- 행렬을 곱함
- $Mx=n$ 의 방정식을 푸는 x 를 찾음

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"
```

3. data.frame

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,]

df[2, 2]

List subsetting

df\$x

df[[2]]

Understanding a data frame

View(df) See the full data frame.

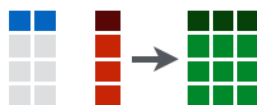
head(df) See the first 6 rows.

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.frame*을 출력
- 처음 6개 행의 *df*를 보고 싶을때 (cf. *tail(df)*)

- *nrow*, *ncol*, *dim*, *head*, *tail*, *cbind*, *rbind*는 대부분 프로그램에 아주 자주 사용하게 되는 명령어입니다.
- 이 명령어를 자주 사용하면 *excel*에서 *R*로 넘어온 두려움을 많이 없앨 수 있습니다.

3. *data.frame*

- *data.frame*을 생성할 때 문자로된 데이터를 자동으로 **factor**로 처리함!
- 따라서 *stringsAsFactors=FALSE*의 *option*을 *data.frame* 생성시 추가 권장 (그리고 *factor* 변수로 원할때에 *as.factor()*를 사용하여 변환)
- 데이터 파일을 불러올 때에도 *stringsAsFactors=FALSE* 옵션을 추가 권장
- 혹은 *options(stringsAsFactors=FALSE)* 명령을 *global option*으로 사용 가능
- 날짜인데 *strin*으로 분류된 변수들에 대해서 *as.Date()*를 이용하여 *type* 변환

4. List

```
sim <- list(numClubs = c(1,8,3,1),  
           car      = c("Mercedes", "Porsche", "Kia"),  
           hobby    = c("golf", "Xfit"))  
  
sim  
  
## $numClubs  
## [1] 1 8 3 1  
##  
## $car  
## [1] "Mercedes" "Porsche" "Kia"  
##  
## $hobby  
## [1] "golf" "Xfit"  
  
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
```

4. List

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.

자료구조 *summary*

The slide features a light gray background with a subtle gradient. In the top-left and bottom-right corners, there are clusters of realistic water droplets of various sizes, rendered with soft shadows and highlights to give them a three-dimensional appearance. The main title is centered on the slide.

Module 4: 프로그램 제어 (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)  
}
```

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)

Return x sorted.

table(x)

See counts of values.

rev(x)

Return x reversed.

unique(x)

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.no(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.

<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.

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.

Variable Assignment

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




The Environment

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

	<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, n)</code> 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.
```

<code>l[[2]]</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.

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

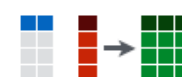
<code>df\$x</code>		<code>df[[2]]</code>	
Understanding a data frame			
<code>View(df)</code>	See the full data frame.		
<code>head(df)</code>	See the first 6 rows.		

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

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

`cbind` - Bind columns.



`rbind` - Bind rows.



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.

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.

Statistics

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

Distributions

	Random Variates	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>punif</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.

실습 과제

- `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] " AAAAAAATA  "  
## [1] "AAAAAAAAA"
```