Basic Syntax

작업 디렉터리

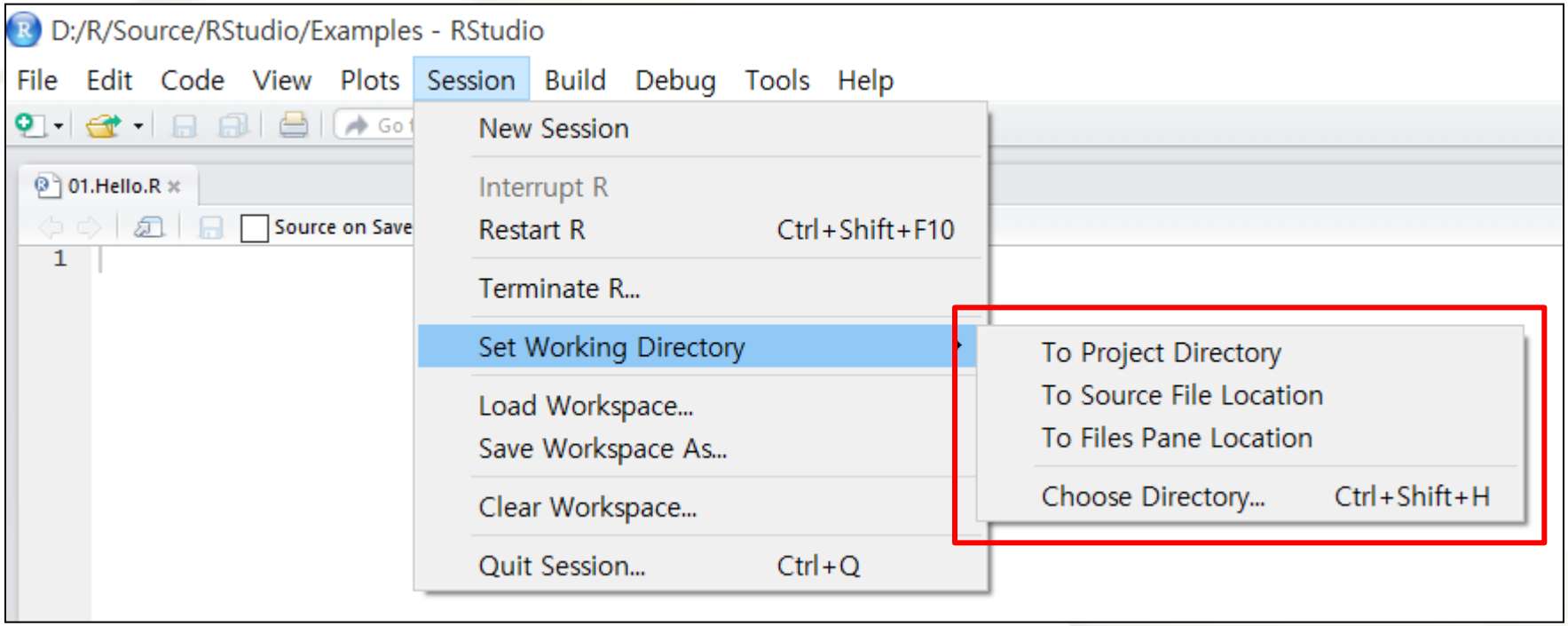
- R이 파일을 읽고 저장하는 기본 경로

- 명령어로 작업 디렉터리 설정

```
setwd("C:/R/Source/RGui")
getwd()
[1] "C:/R/Source/RGui"
```

작업 디렉터리

- RStudio 설정 방법



기본 구문 규칙

- 모든 명령은 대소문자 구분.

```
print("hello")
[1] "hello"

Print("hello")
Error in Print("hello") : could not find function "Print"
```

- 한 줄에 한 개의 명령문 작성.
  - 한 줄에 두 개의 명령문을 작성할 경우 각 문장은 ;으로 구분.

```
> 1 + 2; 2 * 3; 4/2
[1] 3
[1] 6
[1] 2
```

기본 구문 규칙

- 주석
  - 해석기가 해석하지 않는 문장으로 코드 설명을 위해 사용
  - # 으로 표시 (# 부터 그 행의 끝까지 주석)

```
print("hello") # this is comment
[1] "hello"
print("hello") this is comment
Error: unexpected symbol in "print("hello") this"
```

- 문자열
  - 0개 이상의 문자 집합
  - 작은 따옴표 또는 큰 따옴표를 사용해서 표현

```
print('hello')
[1] "hello"
print("hello")
[1] "hello"
print(hello)
Error in print(hello) : object 'hello' not found
```

화면에 출력 – print() 함수 사용

- 형식

```
print(출력 내용)
```

- 사용 사례

```
print(1 + 2)
[1] 3

print('a')
[1] "a“

print(pi)
[1] 3.141593

print(pi, digits = 3)
[1] 3.14
```

화면에 출력 – cat() 함수 사용

- 형식 → cat(출력 내용 리스트)
- 두 개 이상의 데이터를 결합해서 출력 가능
- 사용 사례

```
cat(1, ':', 'a', '\n', 2, ':', 'b', '\n')
1 : a
 2 : b
```

- 자동으로 각 데이터 사이에 공백 삽입
  - sep 전달인자를 사용해서 변경 가능
- 줄 바꿈, 탭 등의 키보드 입력을 표시하기 위해 특수문자 사용

특수 문자	표현 내용	특수 문자	표현 내용
\n	enter	\b	backspace
\r	home	\"	"
\t	tab	\'	'

R 변수

- 데이터를 저장하는 저장 공간 (메모리 상의 저장 공간)
- 데이터를 저장할 때 <-, <<-, = 등의 연산자 사용 (주로 <- 연산자 사용)
- 사용 사례

```
var1 <- 'aaa'
var2 <- 111
var3 <- Sys.Date()
var4 <- c("a", "b", "c")

var1
[1] "aaa"
var2
[1] 111
var3
[1] "2015-07-15"
var4
[1] "a" "b" "c"
```

교재 59 ~ 65 내용 참고

R 변수 명명 규칙

- 대소문자 구분
- TRUE, FALSE, NA, NULL 등과 같은 예약어를 사용할 수 없음
- 문자, 숫자, _, . 사용 가능
- 문자와 .으로만 시작할 수 있으며 .으로 시작할 경우 바로 숫자를 쓸 수 없음
- 명명 사례
  - 올바른 명명 : a, b, a1, a2, .x
  - 잘못된 명명 : .2, _a, 2a

R 변수 목록

- 생성한 모든 변수 확인 – objects(), ls() 함수 사용
  - .으로 시작하는 숨김 변수는 표시되지 않음 → all.names 전달인자를 TRUE로 설정하면 표시 가능

```
objects()
[1] "c"    "date" "name" "num1" "num2" "seq1" "var1" "var2" "var3"

objects(all.names = T)
[1] ".a"   "c"    "date" "name" "num1" "num2" "seq1" "var1" "var2"
"var3"
```

R 변수 제거

- 지정된 변수 제거

```
str1 <- "I'm John Doe"
str1
[1] "I'm John Doe"

rm(str1)
str1
에러: 객체 'str1'를 찾을 수 없습니다
```

- 모든 변수 제거

```
str1 <- 'test'
str2 <- 'test2'
str3 <- 'test3'
objects()
[1] "str1" "str2" "str3"

rm(list = ls())
objects()
character(0)
```

R 자료형

- 데이터의 종류
- 데이터의 저장 형식 또는 사용할 수 있는 데이터의 범위를 구분하는 방법
- 종류 → 연속형과 범주형
- R은 변수에 명시적으로 자료형을 지정하지 않기 때문에 하나의 변수에 여러 자료형의 데이터를 저장할 수 있음
- 사용되는 데이터는 자료형에 의해 구분되며 변수는 저장된 데이터의 자료형을 반영

R 자료형 – 숫자형

- 숫자 자료형
  - 정수, 부동 소수점 데이터 등 연산 가능한 수치 데이터

  - 사용 사례

```
a <- 3
b <- 4.5
c <- a + b
print(c)
[1] 7.5

class(c)
[1] "numeric"
```

R 자료형 – 숫자형

- 숫자형에 사용하는 주요 산술 연산자

연산 기호	설명	사용 사례
+	더하기	5 + 6 → 11
-	빼기	5 – 4 → 1
*	곱하기	5 * 6 → 30
/	나누기 (소수점 이하 연산)	5 / 3 → 1.666667
%/%	나눗셈의 몫 (정수부까지만 연산)	5 %/% 3 → 1
%%	나눗셈의 나머지	5 %% 3 → 2
^, **	거듭제곱	3^2 → 9 3 ** 2 → 9

R 자료형 – 문자형

- 0개 이상의 문자 집합
  - 단일 문자, 문자열 구분 없이 전부 문자열로 처리

- 데이터 표시는 '데이터'또는 "데이터"와 같이 '와 " 사용

- 사용 사례

```
'First'
[1] "First"

'Second'
[1] "Second"

class('1')
[1] "character"

class(1)
[1] "numeric"
```

R 자료형 - 문자형

- 문자형 숫자를 숫자형으로
  - R에서 숫자 2와 "2" 또는 '2'는 다른 방식으로 처리됨

```
2 + 2
[1] 4

"2" + "2"
Error in "2" + "2" : 이항연산자에 수치가 아닌 인수입니다
```

  - R은 문자형을 숫자형으로 변형해 주는 as.numeric() 함수 제공

```
as.numeric("2") + as.numeric("2")
[1] 4
```

R 자료형 - 논리형 (TRUE / FALSE)

- 논리형 데이터
  - 참과 거짓을 표현하는 데이터 → TRUE / T 또는 FALSE / F 로 표현
  - 숫자 0은 FALSE로 나머지는 TRUE로 해석

- 논리 연산자 (논리형 데이터에 사용하는 연산자)

연산 기호	설명	사용 사례
&	피 연산항이 모두 TRUE이면 TRUE, 아니면 FALSE	TRUE & TRUE → TRUE TRUE & FALSE → FALSE FALSE & TRUE → FALSE FALSE & FALSE → FALSE
|	피 연산항이 모두 FALSE이면 FALSE, 아니면 FALSE	TRUE | TRUE → TRUE TRUE | FALSE → TRUE FALSE | TRUE → TRUE FALSE | FALSE → FALSE
!	피 연산항의 반대 논리 값	!TRUE → FALSE !FALSE → TRUE

R 자료형 – 논리형 (TRUE / FALSE)

- 사용 사례

```
# 논리형 변수 선언 및 값 할당
t1 <- TRUE
t2 <- T
t3 <- 1
f1 <- FALSE
f2 <- F
f3 <- 0
```

```
# 논리 연산 수행
t1 & f1
[1] FALSE

t1 & t2
[1] TRUE

f1 & f2
[1] FALSE

t3 | f3
[1] TRUE

t3 | t1
[1] TRUE

f3 | f2
[1] FALSE
```

R 자료형 - NA 타입

- NA 타입
  - 유효하지 않은 값 (not available, 결측치)
  - 변수의 값이 NA인지 확인하기 위해 is.na(변수명) 함수 사용
  - 일반적으로 명시하지 않을 경우 연산에 포함되며,
  - NA 값을 연산에서 제거하기 위해 na.rm인자를 TRUE로 설정

- 사용 사례

```
cat(1, NA, 2, "\n")
1 NA 2

sum(1, NA, 2)
[1] NA

sum(1, NA, 2, na.rm = T)
[1] 3
```

NULL

- 값이 존재하지 않는 상태 (not exist)
- 변수의 값이 NULL인지 확인하기 위해 is.null(변수명) 함수 사용
- 연산에 포함되지 않음
- 사용 사례

```
cat(1, NULL, 2, "\n")
1 2

sum(1, NULL, 2)
[1] 3
```

R 자료형 – Factor 형

- 범주형 변수 (Categorical Variable)
  - 값이 대상을 분류하는 의미를 지니는 변수
  - 숫자를 기반으로 만든 경우에도 크기를 의미하지 않기 때문에 산술 연산을 적용할 수 없음
  - R에서는 Factor 형으로 범주형 데이터 표현

- 사용 사례

```
xv <- c("A", "C", "B", "A", "A", "C", "B")
xv
[1] "A" "C" "B" "A" "A" "C" "B"

x <- factor(xv, levels = c('A', 'B', 'C'))
x
[1] A C B A A C B
Levels: A B C

y <- factor(xv, levels = c('A', 'B')) # 'A', 'B'가 아닌 것은 결측치
y
[1] A    <NA> B    A    A    <NA> B
Levels: A B
```

R 자료형 – Factor 형

- 목록의 개수는 nlevels, 목록의 내용은 levels 함수를 사용해서 조회

```
gender <- factor('m', c('m', 'f'))
nlevels(gender)
[1] 2

levels(gender)
[1] "m" "f"
```

- 목록 내용 변경

```
levels(gender) <- c('male', 'female')
gender
[1] male
Levels: male female
```

R 자료형 – 날짜와 시간

- 날짜만 다루는 형식과 날짜와 시간을 함께 다루는 형식 제공.

- 현재 날짜, 시간 확인 함수 사용 사례

```
Sys.Date()
[1] "2018-05-01"

Sys.time()
[1] "2015-07-14 23:47:24 KST"

date() # 문자열 반환
[1] "Tue Jul 14 23:47:28 2015"
```

- 문자열 → 날짜 형식 변환

```
as.Date('2014-11-01')
[1] "2014-11-01"

as.Date('01-11-2014') #error
[1] "0001-11-20"
```

R 자료형 – 날짜와 시간

- as.Date 함수의 format 전달인자에서 사용하는 서식

형식	설명	형식	설명
%d	숫자 형식의 날짜	%b	월을 축약어로
%m	숫자 형식의 월	%B	월을 전체 이름으로
%y	두 자리 숫자 형식의 년도	%a	요일을 축약어로
%Y	네 자리 숫자 형식의 년도	%A	요일을 전체 이름으로

- 사용 사례

```
as.Date('2014년 11월 1일', format = '%Y년 %m월 %d일')
[1] "2014-11-01"

as.Date('01112014', format = '%d%m%Y')
[1] "2014-11-01"

as.Date('011114', format = '%d%m%y')
[1] "2014-11-01"
```

R 자료형 – 날짜와 시간

- as.Date 함수를 이용한 날짜 연산

```
as.Date(10, origin="2014-11-10")
[1] "2014-11-20"

as.Date(-10, origin="2014-11-10")
[1] "2014-10-31"

as.Date("2014-11-30") - as.Date("2014-11-01")
Time difference of 29 days

as.Date("2014-11-01") + 5
[1] "2014-11-06"

as.Date('2014-11-01 20:00:00') - as.Date('2014-11-01 18:30')
Time difference of 0 days #시간 식별 못함
```

R 자료형 – 날짜와 시간

- POSIXlt와 POSIXct를 사용할 경우 시간 까지 연산 가능
  - POSIXlt는 날짜를 년, 월, 일로 표시하는 리스트형
  - POSIXct는 1970년을 기준으로 초 단위 시간 계산

```
as.POSIXct("2016-07-01 20:00:00") - as.POSIXct("2016-07-01 18:30:30")
Time difference of 1.491667 hours
```

제어문

- 프로그램의 실행 흐름을 제어하는 구문

종류	구문
선택문	if - else : 단순 선택문 및 다중 선택문 구성 ifelse : 양자 택일형 선택문 구성
반복문	while : 반복의 횟수를 예측하지 어려운 경우 for : 반복의 횟수가 명확한 경우

- 제어문에서 자주 사용되는 연산자

연산자	의미	연산자	의미
==	같다	!	논리부정 - 반대 논리 값
!=	다르다	& (&&)	논리 곱
>=	크거나 같다	| (||)	논리 합
>	크다		
<=	작거나 같다		
<	작다		

제어문

- if – else 구문 사용 사례

```
x = 10
if (x < 0) print(-x)
else print(x)
[1] 1

x = -10
if (x < 0) print(-x)
else print(x)
[1] 1
```

```
x = 1
if (x > 0) print(x * 2)
else if (x == 0) print(0)
else print(x * 3)
[1] 2

x = 0
if (x > 0) print(x * 2)
else if (x == 0) print(0)
else print(x * 3)
[1] 0

x = -1
if (x > 0) print(x * 2)
else if (x == 0) print(0)
else print(x * 3)
[1] -3
```

제어문

- ifelse 구문 사용 사례

```
no <- 3
ifelse(no %% 2 == 0, 'even number', 'odd number')
[1] "odd number"
```

- while 구문 사용 사례

```
x <- 1
while (TRUE) {
  x <- x + 1
  if (x == 4) break;
  print(x)
}
[1] 2
[1] 3
```

```
x <- 0
while (TRUE) {
  x <- x + 1
  if (x == 6) break
  if (x %% 2 == 0) next
  print(x)
}
[1] 1
[1] 3
[1] 5
```

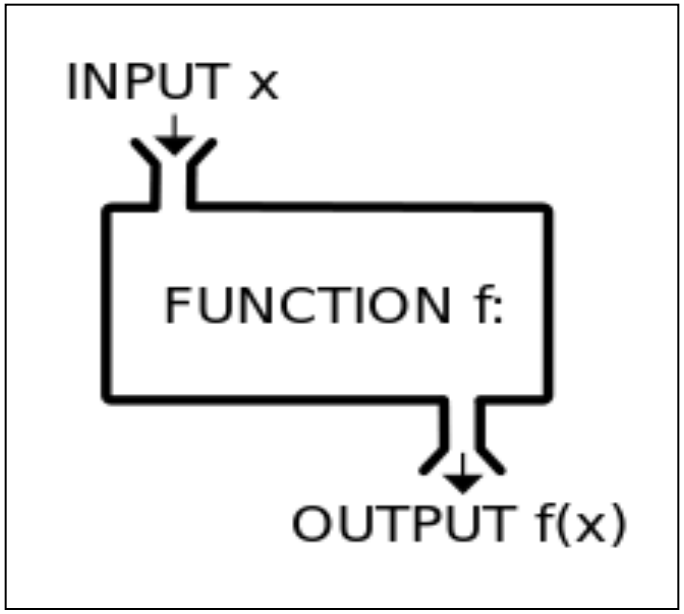
제어문

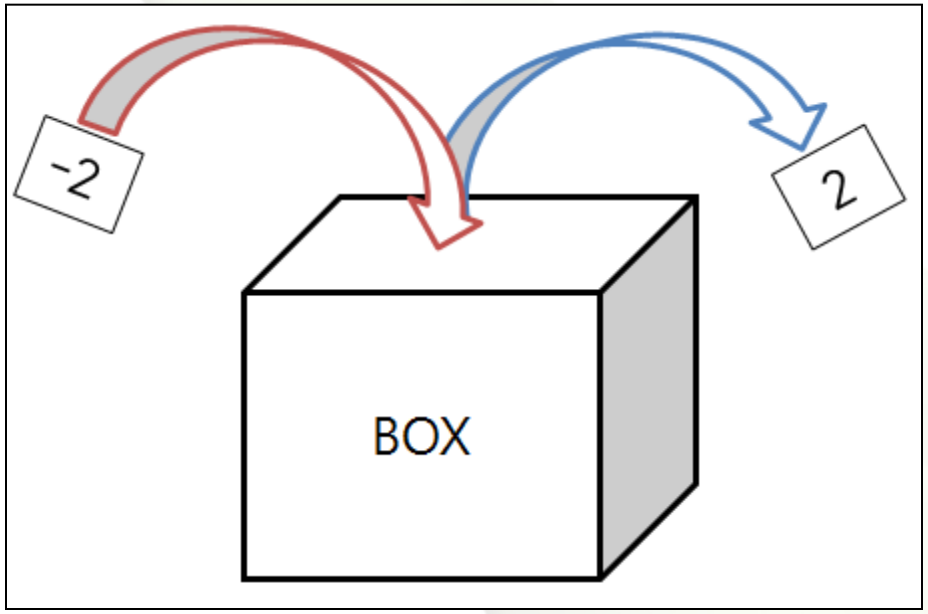
- for 구문 사용 사례

```
x <- 10
for (i in 1 : x) {
  print(i)
}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

함수

- 특정 기능을 수행하는 실행 명령문 (코드) 집합
- 코드를 재사용하는 단위
- 함수에 어떤 값을 입력하면 함수가 실행된 후 결과를 출력
  - 함수에 입력되는 값 → 전달인자 또는 파라미터
  - 함수가 반환하는 값 → 반환 값





함수

- R은 데이터 분석에 필요한 다양한 함수를 만들어서 제공
- R이 제공하는 기본 함수에 포함되지 않은 다른 함수들도 다양한 패키지를 통해 사용 가능
- 함수 사용 → 함수 호출

```
> score <- c(85, 90, 76, 82, 79)

> min(score) # 평균을 구하는 함수
[1] 76

> max(score) # 최대 값을 구하는 함수
[1] 90

> summary(score) # 요약 기초 통계 값을 구하는 함수
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   76.0    79.0    82.0    82.4    85.0    90.0
```

사용자 정의 함수

- 정의 형식

```
함수이름 <- function(전달인자 목록) {
        실행문1
        실행문2
        ...
        return (반환 값) #리턴 되는 반환 값은 반드시 ()로 표시
}
```

- 전달인자 : 함수를 호출하는 쪽에서 함수로 데이터를 제공하는 방법
  - 전달인자에 기본 값 적용 가능
  - 기본값이 지정된 경우 호출할 때 전달 인자 생략 가능
  - 전달 인자의 이름을 명시해서 함수 호출 가능

- 반환 값 : 함수에서 함수를 호출한 쪽으로 데이터를 제공하는 방법
  - 반환 값은 반드시 ()로 표시

사용자 정의 함수

- 사용 사례

```
# 함수 정의

myfunc1 <- function() {
  return (10)
}
myfunc1()
[1] 10

myfunc2 <- function(a, b) {
  return (a * b)
}
myfunc2(10, 20)
[1] 200

myfunc3 <- function(a, b = 10) {
  return (a * b)
}
```

```
# 함수 호출

myfunc3(10, 20)
[1] 200

myfunc3(10)
[1] 100

myfunc3(b = 1, a = 2)
[1] 2
```

패키지 관리

- 특정한 기능을 수행하는 코드들의 집합

- R에서 어떤 작업을 하려면 해당 기능을 제공하는 패키지 필요
  - R을 설치할 때 같이 설치되는 기본 패키지와
  - 필요에 따라 추가로 설치 할 수 있는 확장 패키지 제공

- 패키지 설치

```
install.packages("ggplot2")
install.packages(c("dplyr", "ggplot2"))
```

- 작업 공간으로 패키지 로딩

```
library(ggplot2)
```

패키지 관리

- 패키지 업데이트

```
update.packages("ggplot2")
update.packages()
```

- 패키지 설치 경로 확인

```
.libPaths()
```

- 설치된 패키지 목록

```
installed.packages()
```

- 패키지 삭제

```
remove.packages("dplyr")
```

- 패키지 정보 확인

```
library(help = ggplot2)
```