R Programming Language

R 산술, 논리 연산

*R*은 계산에 능합니다

R 산술 연산자
: Arithmetic Operators

▶ 산술 연산자 : 산술 계산을 할 때 사용하는 기호

산술 연산자	기능
+	더하기
-	빼기
*	곱하기
/	나누기
^, **	제곱
%/%	나눗셈의 몫
%%	나눗셈의 나머지

```
> 7 + 5 # 덧셈
[1] 12
> 7 - 5 # 뺄셈
[1] 2
> 7 * 5 # 곱셈
[1] 35
> 7 / 5 # 나눗셈
[1] 1.4
> 7 ^ 5 # 제곱
[1] 16807
> 7 ** 5 # 제곱
[1] 16807
> 7 %/% 5 # 나눗셈의 몫
[1] 1
> 7 %% 5 # 나눗셈의 나머지
[1] 2
```

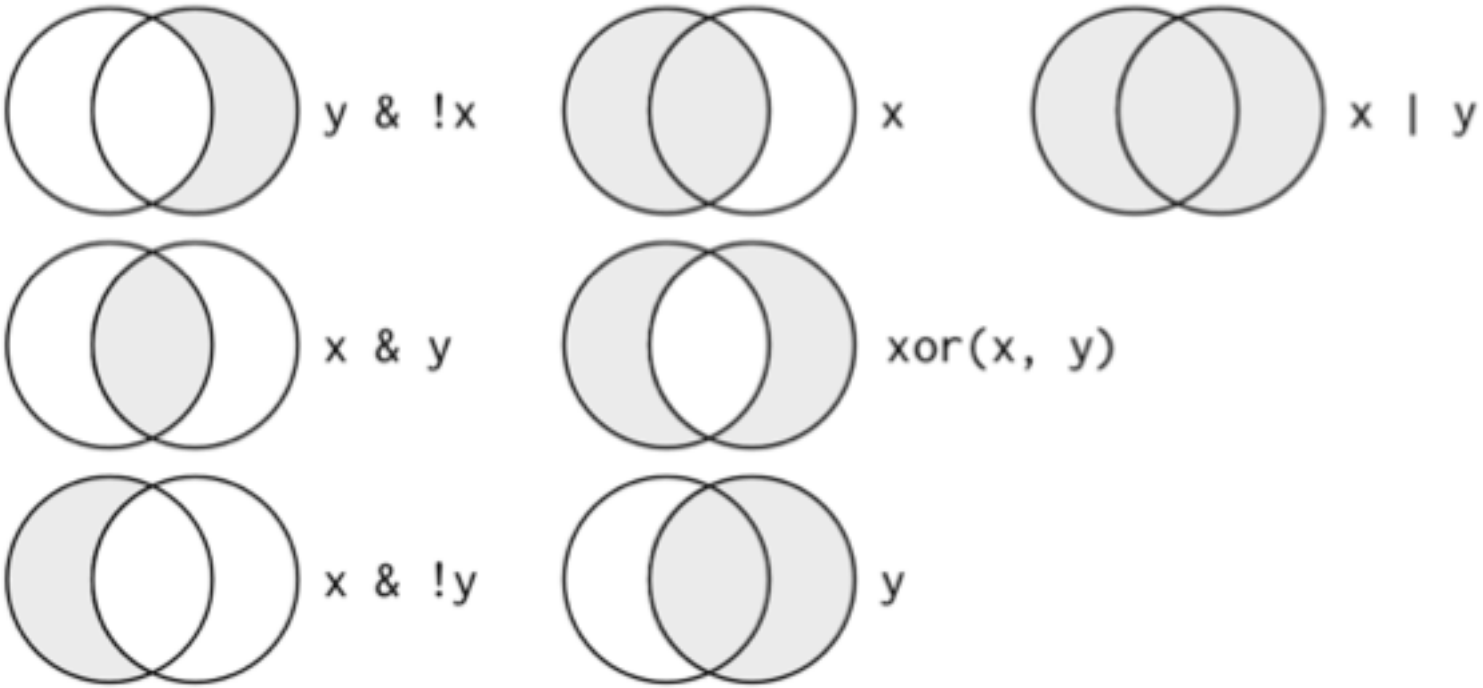
R 논리 연산자
: Logical Operators

- 논리 연산자 : 조건 비교, 지정을 할 때 사용

논리 연산자	기능
<	작다
<=	작거나 같다
>	크다
>=	크거나 같다
==	같다
!=	같지 않다
|	또는
&	그리고
!	부정*(not)*
%in%	매칭 확인

```
> 5 < 6
[1] TRUE
> 5 > 6
[1] FALSE
> 5 == 6
[1] FALSE
> 5 != 6
[1] TRUE
> 5 > 6 | 8 > 7 # OR 연산
[1] TRUE
> 5 > 6 & 8 > 7 # AND 연산
[1] FALSE
> 5 %in% c(1, 2, 3, 4, 5)
[1] TRUE
```

TRUE or FALSE



논리값은 ***TRUE*** 혹은 ***FALSE***로 반환되며
줄여서 ***T*** 혹은 ***F***로 사용할 수 있다

R 도구 준비

변수, 객체, 함수 그리고 패키지

변수 *(Variable)*
: 통계의 입장

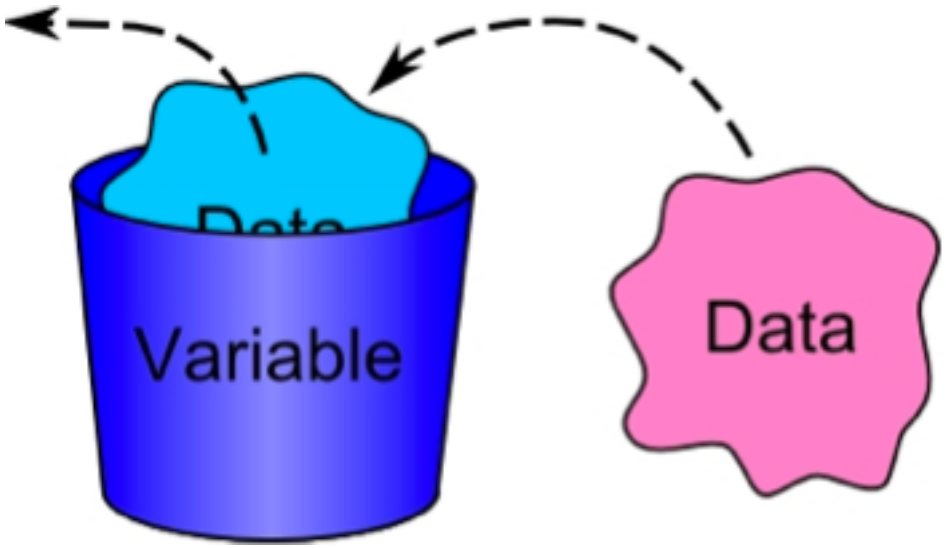
- 다양한 값을 담고 있는 '변하는 수'

- 변수는 데이터 분석의 대상
  - 데이터 분석이란 변수 간에 어떤 관계가 있는지 파악하는 작업

- <-> 상수
  - 상수는 변하지 않는 값으로 되어 있는 속성
  - 상수는 변수와 달리 분석 대상이 아님

id	*class*	*math*	*english*	*science*
1	*1*	*50*	*98*	*50*
2	*1*	*60*	*97*	*60*
3	*1*	*45*	*86*	*78*
4	*1*	*30*	*98*	*58*
5	*2*	*25*	*80*	*65*
6	*2*	*50*	*89*	*98*
7	*2*	*80*	*90*	*45*
8	*2*	*90*	*78*	*25*
9	*3*	*20*	*98*	*15*
10	*3*	*50*	*98*	*45*

변수 *(Variable)*
: 프로그래밍 언어의 입장



- 데이터를 담는 그릇, 혹은 상자
- 변수에 넣은 데이터는 언제든 불러올 수 있고, 경우에 따라 자유롭게 값을 변경할 수 있다
- ***R***은 통계를 주로 다루는 패키지이므로 변수***(Variable)***라는 명칭은 통계에 양보하고 객체***(Object)***라는 용어로 사용하기도 한다

객체 생성
: 생성 규칙과 할당

- 객체명 생성 규칙
  - 문자, 숫자, 언더바*(_)*, *dot(.)*을 조합해 정할 수 있다
  - 단, 문자로 시작해야 한다
  - 변수명은 대소문자를 구분하므로 주의
- 할당
  - 객체에 값을 할당할 때는 할당 연산자 *(<-, ->)*를 사용
  - <- 할당 연산자는 = 로 사용해도 무방
- 삭제
  - *rm()* 함수를 이용

```
> eng <- 90
> 80 -> math
> total = eng + math
> total
[1] 170
```

<- 는 = 로 사용해도 무방

{변수명} <- {값}

{값 혹은 수식 처리} -> {변수명}

여러 값으로 구성된 객체
: Vector

- ***Vector***
  - 동일한 데이터 유형*(*숫자 또는 문자 등*)*의 단일 값들이 일차원적으로 구성된 것
  - *c()* 함수를 이용하면 여러 값으로 구성된 벡터를 생성할 수 있음
    - 괄호 안에 콤마*(,)*를 이용해 값을 나열
  - 연속된 값으로 벡터를 구성할 때
    - *{*시작값*}:{*종료값*}*
    - *seq({*시작값*}, {*종료값*})* 함수
    - 증가값을 부여하고자 할 때는 *seq* 함수의 *by* 파라미터를 이용

```
> var1 <- c(2, 4, 6, 8, 10)
> var2 <- 1:10
> var3 <- seq(1, 10)
> var1
[1]  2  4  6  8 10
> var2
[1]  1  2  3  4  5  6  7  8  9 10
> var3
[1]  1  2  3  4  5  6  7  8  9 10
> var4 <- seq(2, 10, by = 2)
> var4
[1]  2  4  6  8 10
```

다양한 객체의 타입들
: *R* 언어의 자료형

Data Type	의미	값
numeric	실수	1, 12.3
integer	정수	1L, 12L
complex	복소수	2 + 3i
character	문자	"r", "programming"
logical	논리	TRUE, FALSE, T, F
factor	명목/순서형	1, 2, a, b
Date	날짜	"2017-11-24", "24/11/17"

수치형

기본 자료형

객체의 자료형 확인
: *is*

- 객체에 자료형을 확인하기 위해서는 ***is()*** 함수를 사용

```
> v <- c(1, 2, 3, 4, 5)
> is(v)
[1] "numeric" "vector"
```

- 기본적으로 ***R***은 정수, 실수 구분하지 않고 ***numeric***으로 취급
  - 명시적으로 정수형임을 구분하려면 숫자 뒤에 ***L***을 붙여 표기
- 좀 더 세부적으로 자료형을 확인하고자 한다면 ***is.{type}()*** 함수를 이용

```
> is.numeric(v)
[1] TRUE
> is.vector(v)
[1] TRUE
> is.integer(v)
[1] FALSE
```

객체의 자료형 변환
: *as*

- 객체에 자료형을 변환하기 위해서는 ***as.{type}()*** 함수를 사용

```
> v
[1] 1 2 3 4 5
> is(v)
[1] "numeric" "vector"

> v <- as.integer(v)
> is(v)
[1] "integer"              "numeric"              "vector"
[4] "data.frameRowLabels"
> is.integer(v)
[1] TRUE
```

- 일반적은 산술연산에서는 정수형을 구분할 필요는 없으나 대량의 데이터를 다룰 때에는 ***integer***와 ***numeric***은 반드시 구분 **->** ***integer***의 연산이 더 빠르다

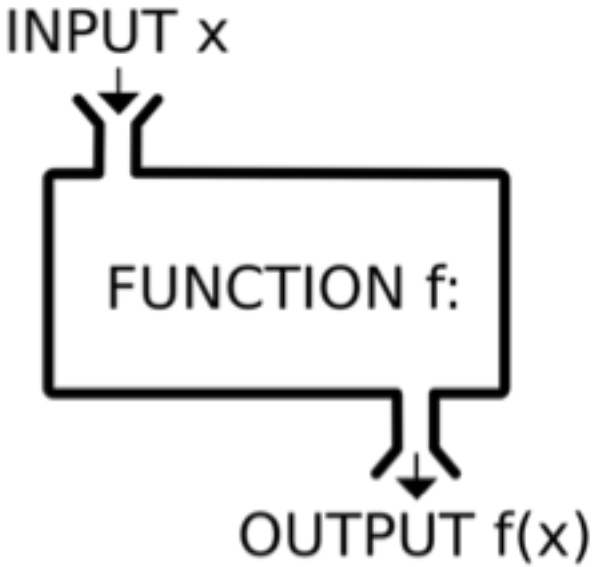
특수한 데이터 타입들

유형	의미
NA	결측값(*missing value*) = 값이 존재하지 않음
NULL	데이터 유형과 자료의 길이도 ***0***인 비어있는 값
NaN	수학적으로 정의가 불가한 수
Inf	무한값(*infinite*)

- 이러한 유형의 데이터들은 통계 함수에 오류를 발생시키거나 잘못된 결과를 유발
- 따라서 통계 함수를 실행시킬 때는 이들 값을 어떻게 처리할 것인지 고려하고 처리해 주어야 함

함수 *(Function)*

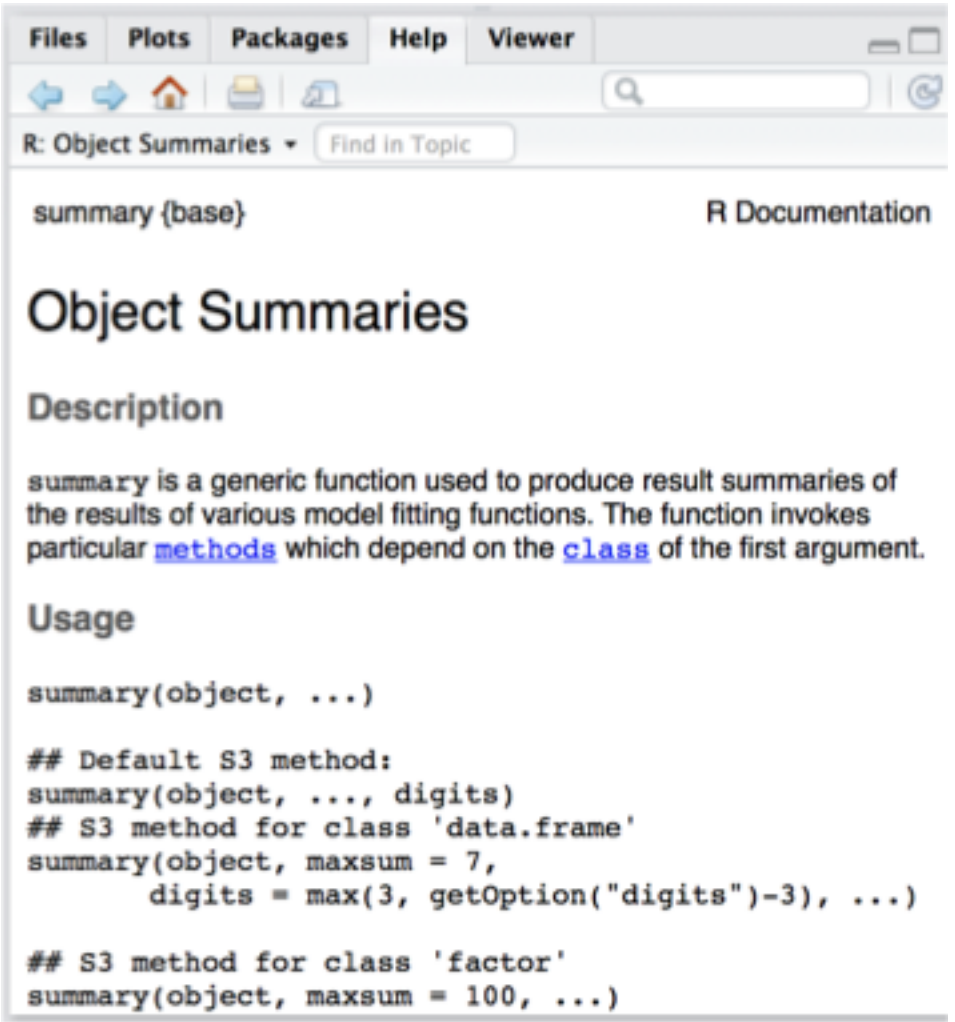
- 입력 값을 가지고 어떤 일을 수행한 다음 그 결과물을 내 놓는 것
- 함수를 사용하는 이유
  - 반복되는 부분이 있을 경우 재활용을 위해
  - 프로그램의 흐름을 일목요연하게 볼 수 있다
- 함수 중에는 인풋 값이 없는 함수도 있고 아웃풋 값이 없는 함수도 있다



함수 기능이 궁금할 때
: help 혹은 *?*

- 콘솔에서 다음과 같이 입력하면 함수의 사용법을 알 수 있다
  - *help({함수명})*
  - *?{함수명}*
- *R*의 함수는 방대하고, 함수마다 다양한 파라미터가 있으므로 모든 함수를 외우려 하면 안된다
  - 도움말을 참고해 가면서 자주 사용하다 보면 점차 익숙해질 것

```
> help(summary)
> ?summary
```



내장 함수 맛보기
: 수치를 다루는 함수들

- ***sum*** **:** 주어진 인수 값들을 모두 더하는 함수
- ***mean*** **:** 주어진 인수 값들의 평균을 구하는 함수
- ***min, max*** **:** 주어진 인수 값들의 최소값, 최대값을 구하는 함수
- ***sample*** **:** 표본 추출 함수
  - ***size*** **:** 추출할 표본의 크기
  - ***replace = T*** **:** 복원 추출 허용

```
> sum(1, 2, 3, 4, 5)
[1] 15
> mean(c(1, 2, 3, 4, 5))
[1] 3
> sample(1:45, size = 6, replace = F)
[1]  7 41 11 39 10 44
```

```
> nums <- c(1, 9, 4, 5, 7, 3)
> max(nums)
[1] 9
> min(nums)
[1] 1
```

반환 값이 있는 함수는 객체에 그 결과값을 저장할 수 있다

내장 함수 맛보기
: 문자열 함수

- ***paste*** **:** 벡터를 문자열로 변환하여 합치는 함수
  - ***paste0*** **:** ***paste*** 함수와 동일하나 ***sep*** 파라미터가 없는 함수

- 파라미터***(parameter)***
  - 함수의 옵션을 설정하는 값
  - '매개변수'라고도 함

- 사용하고자 하는 함수에 어떤 파라미터가 있는지, 그 값의 변화에 따라 어떻게 결과가 반영되는지 잘 살펴보는 것을 권장
  - 도움말을 잘 활용하자

```
> paste("Hello", "R", "Programming")
[1] "Hello R Programming"
> paste("Hello", "R", "Programming", sep = ":")
[1] "Hello:R:Programming"

> paste0("A", c(1, 2, 3, 4, 5))
[1] "A1" "A2" "A3" "A4" "A5"
```

기초 산술 함수

함수	설명	사용예
sin	삼각함수: 사인	sin(pi / 3)
cos	삼각함수: 코사인	cos(pi / 3)
tan	삼각함수: 탄젠트	tan(pi / 3)
abs	절대값	abs(3), abs(-3)
round	반올림	round(56.78), round(56.789, 2)
floor	버림	floor(56.78)
ceiling	올림	ceiling(56.78)
factorial	팩토리얼	factorial(9)

*pi*는 *R* 내장 수치

사용자 정의 함수

- 사용자 함수의 정의

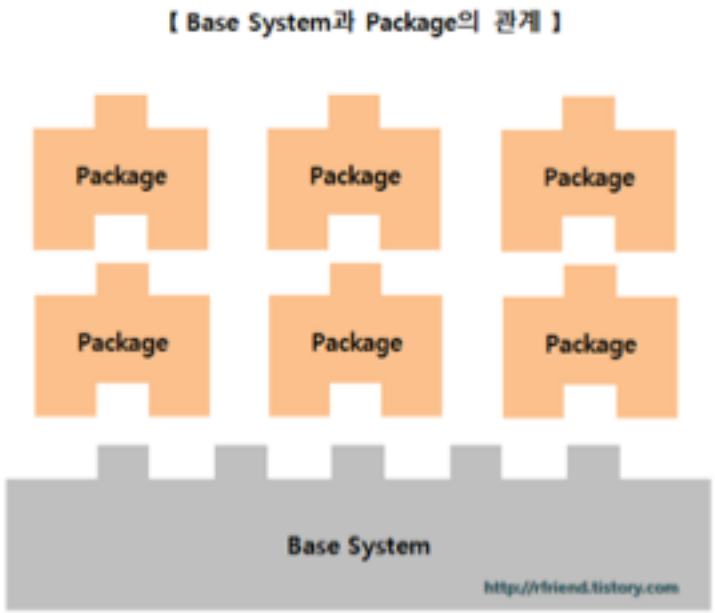
```
{함수명} <- function(함수 인자들) {
    # 수행할 내용들
    return {반환할 값}
}
```

```
mystat <- function(x) {
  return(c(max(x), min(x), sum(x), mean(x)))
}
```

```
> mystat(c(90, 80, 95, 75))
[1]  95  75 340  85
```

패키지 *(Package)*

- 다양한 함수와 기능을 한데 묶어둔 꾸러미
- 특정 기능을 이용하려면 먼저 패키지를 설치해야 한다
- 앱이나 플러그인을 설치하듯 골라 설치할 수 있다



패키지 설치 및 사용

다음 패키지들을 설치해 봅시다

- **ggplot2**
- **plotly**
- **dplyr**
- **reshape2**


```
> # ggplot2 패키지 설치
> install.packages("ggplot2")
```

```
> # ggplot2 패키지 로드
> library(ggplot2)
```

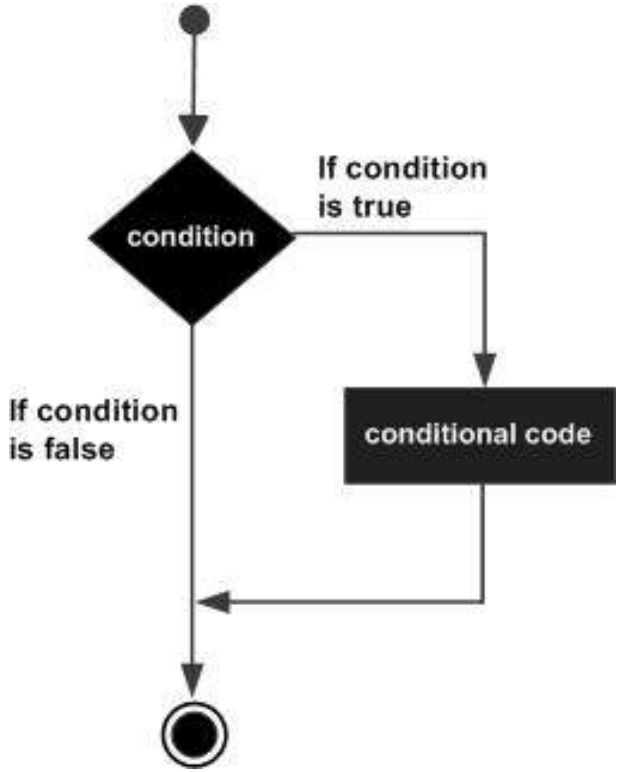
```
> # 변수 설정
> x <- c("a", "b", "c", "b", "a", "e")
> # 빈도 막대 그래프 출력
> qplot(x)
```

흐름 제어 ***(Flow Control)***

조건문과 반복문

조건문
: *if ~*

- 특정 조건에 따라 프로그램의 흐름을 바꿀 수 있다
- 조건절의 내용을 충족하면 특정 내용을 실행한다

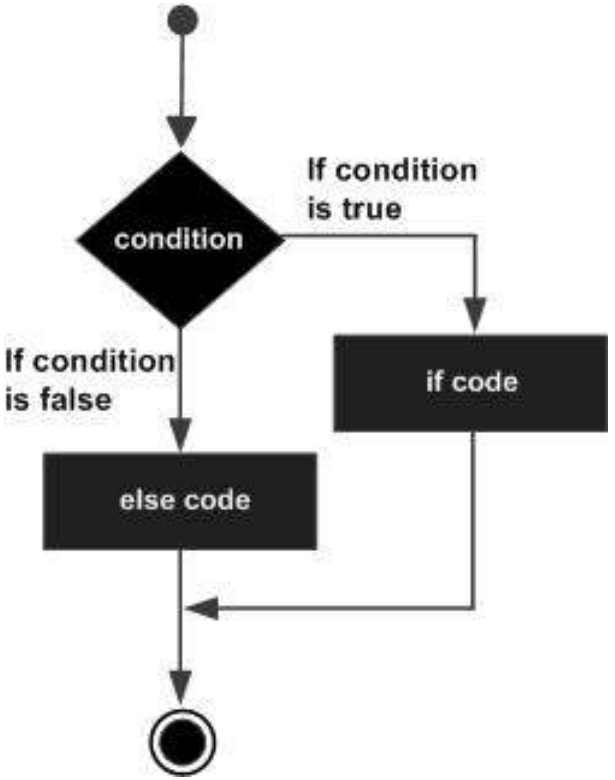



```
if ({조건절}) {
    조건이 참일때의 처리
}
```

조건문
: if ~ else ~

- ***if*** 문의 조건이 충족되지 않을 때 ***else*** 안의 내용이 실행된다
- 점검해야 할 조건이 여러개일 때, ***else if*** 블록으로 여러 개를 넣을 수 있다

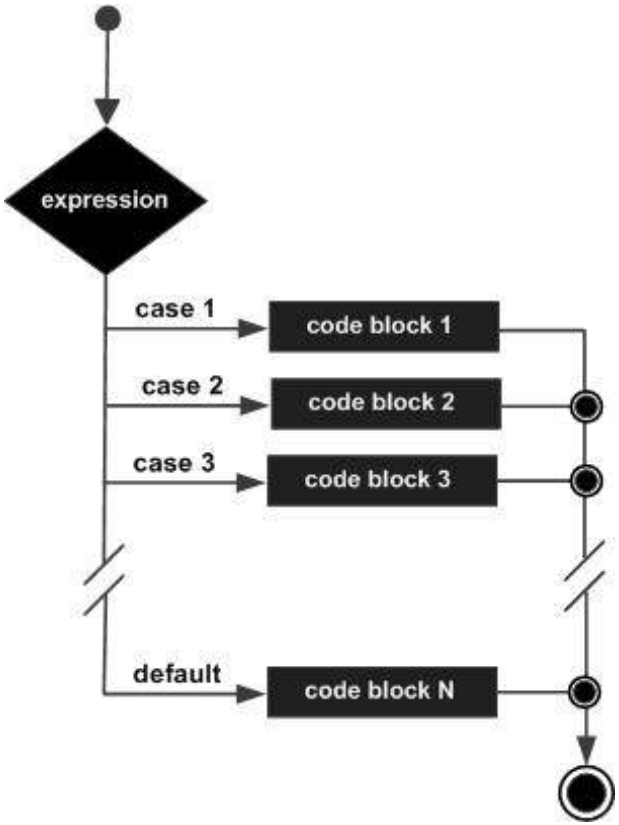
```
if ({조건절}) {
    조건이 참일때의 처리
} else {
    조건이 거짓일 때의 처리
}
```



조건문
: switch

- 조건절의 결과값에 따라 일치하는 ***case*** 값을 돌려줌
- 조건이 일치하지 않았을 때의 처리 ***(default)***는 하지 않음

```
switch ({조건절},
    {when-case1},
    {when-case2},
    {when-case3},
    ...
}
```



조건문
: ifelse

- 지정한 조건에 맞을 때와 맞지 않을 때 각기 다른 값을 반환
- ***R***에서 가장 자주 사용하는 조건문

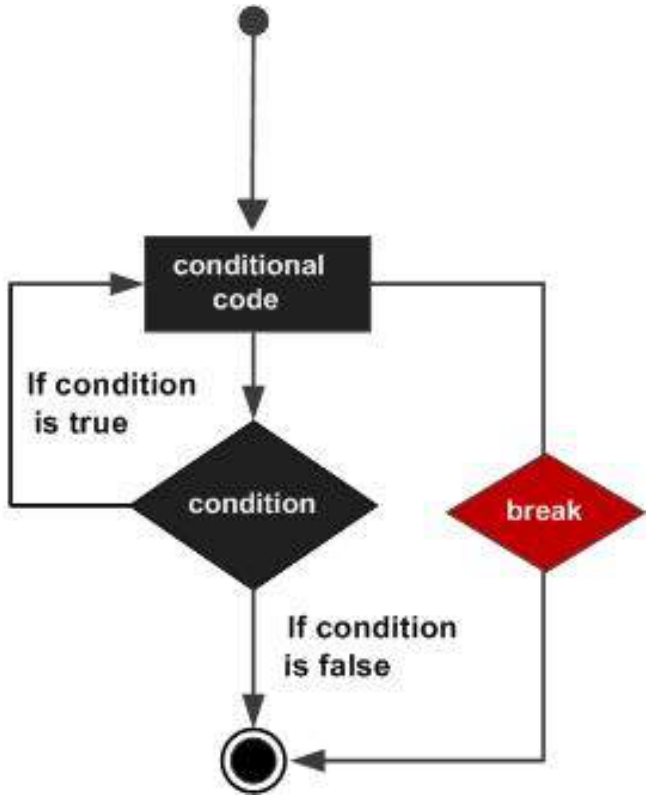
- ***ifelse***를 중첩하면 ***if ~ else if ~ else ~*** 문을 사용했을 때와 같은 복잡한 조건문도 만들 수 있다

```
ifelse({조건}, {참일때의 반환값}, {거짓일때의 반환값})
```

반복문
: repeat

- 동일 구문을 조건이 충족될 때까지 반복 수행
- 최소 *1*회는 무조건 실행

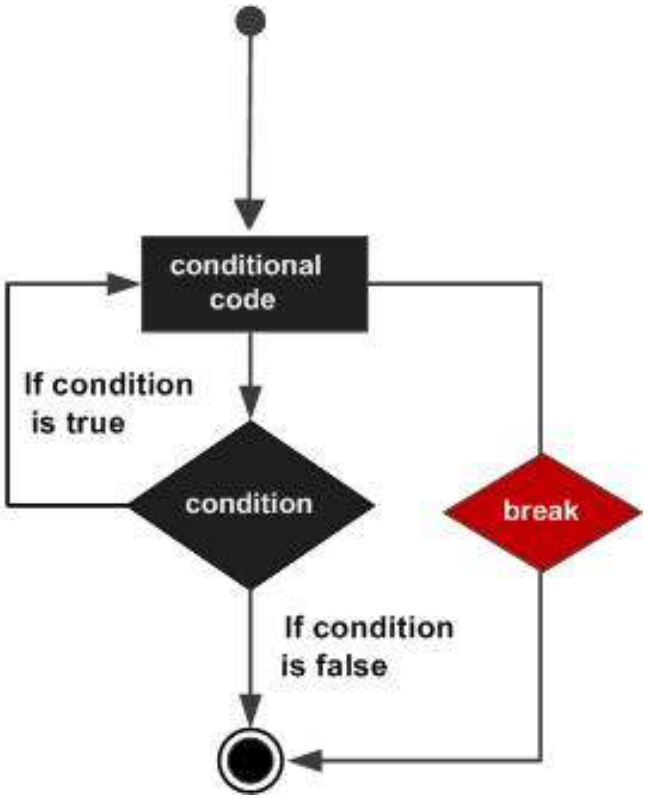
```
repeat {
    {반복구문}
    ...
    if ({조건절}) {
        break
    }
}
```



반복문
: while

- 주어진 조건이 참인 동안 동일 구문을 반복 수행
- 조건 점검을 코드 실행 이전에 하므로 단 한번도 실행되지 않을 수도 있음

```
while ({조건절}) {
    {반복구문}
    ...
}
```



반복문
: *for*

- 특정 횟수만큼 실행해야 하는 루프를 효율적으로 작성할 수 있는 반복 제어 구조

- 단순히 정수나 지정된 숫자에 국한되지 않고 문자 벡터, 논리 벡터, 리스트 등 다양한 자료형 및 표현식으로부터 값을 받아올 수 있음

```
for ({값} in {Vector}) {
    {반복구문}
    ...
}
```