R의 기초

R의 기본 사용법

- 기초 사용법
  - R Studio프로그램에서는 스크립트를 이용해 명령어를 실행, 저장하도록 함 R Studio에서 제공하는 스크립트를 기준으로 설명

R의 기본 사용법

- #(해시 기호)
  - 주석(Comment)의 기능 프로그램 전반적인 내용, 명령어의 내용 등이 무엇을 의미하는지 알 수 있도록 사용자가 설명을 달아주는 기능
  - # 뒤에 있는 한 줄이 주석으로 처리되며, R에서 지정된 문법을 검사하지 않음
  - 단, 한 줄만 주석으로 처리되므로 다른 줄도 하고 싶으면 해당 줄 앞에 #를 써야 함

- ;(세미콜론)
  - 하나의 명령어가 끝났음을 알려주는 기능
  - 한 줄에 하나의 명령밖에 없으면 세미콜론을 해 주지 않아도 명령어가 끝났음을 인식

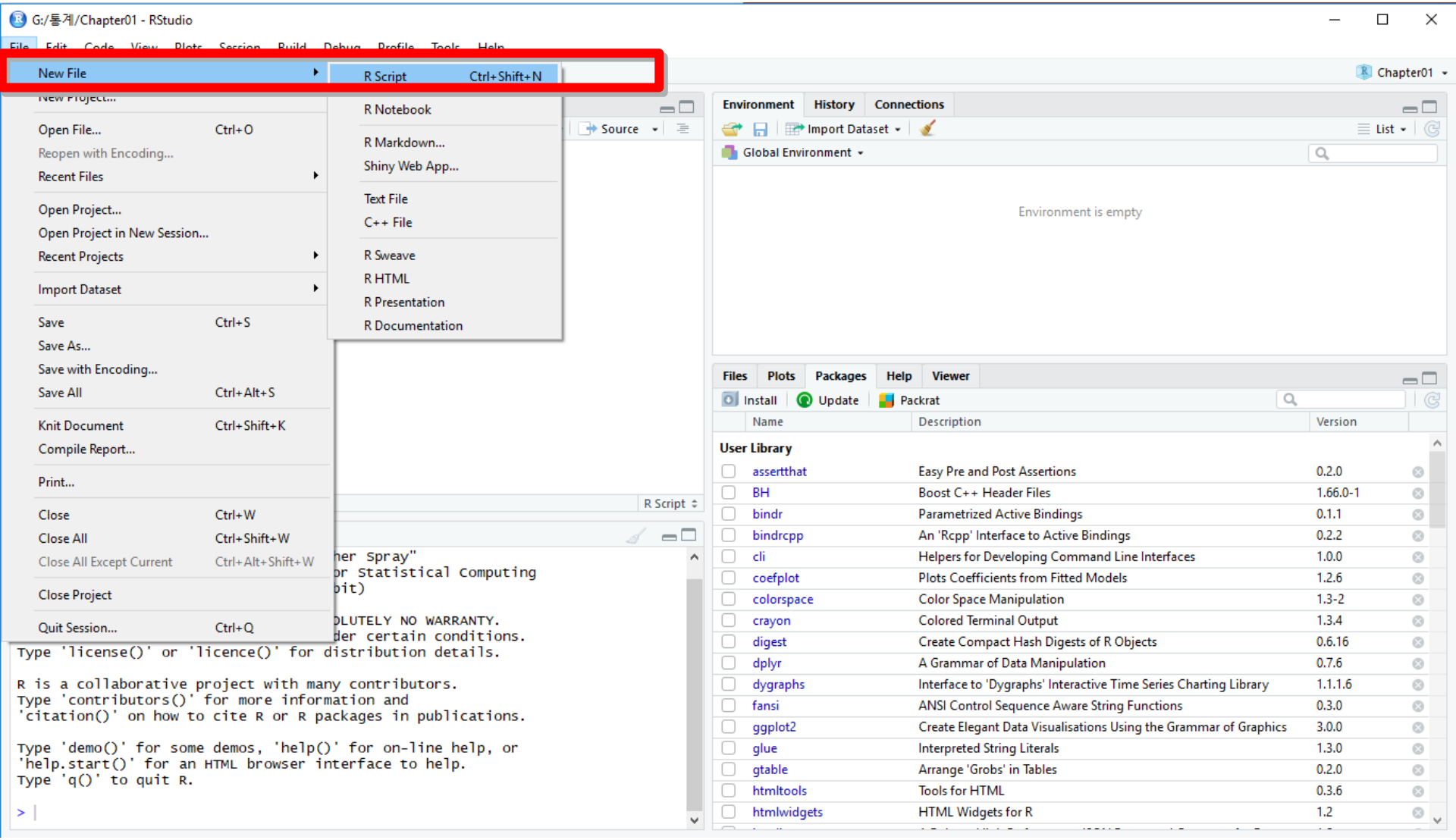
R의 기본 사용법

- Enter(엔터)
  - 다음 줄로 이동할 때 사용
- Ctrl + Enter(컨트롤+엔터)
  - R의 명령어를 실행하는 기능
  - 명령어가 한 줄인 경우 마우스의 위치는 해당 줄의 아무 곳에 있어도 상관이 없음
  - 명령어가 두 줄 이상인 경우 반드시 해당 명령어가 있는 곳을 블록을 잡고 실행
- Shift + Enter(시프트 + 엔터)
  - 동일한 위치에 다른 Argument(함수에 들어가는 값)를 올 수 있도록 해줌
  - Shift + Enter를 이용해 명령어가 길게 표현되는 것을 방지함

R의 기본 사용법

- 대소문자
  - R은 대소문자를 구별(Case Sensitive)
  - 소문자 'x'와 대문자 'X'는 전혀 다른 것을 의미하기 때문에 주의해야 함

R Script 실행하기



R의 연산자

- 산술 연산자(Arithmetic Operator)
- 할당 연산자(Allocation Operator)
- 비교 연산자(Relational Operator)
- 논리 연산자(Logical Operator)

산술 연산자(Arithmetic Operator)

- 1개 이상의 수치에 대한 연산
- 산술 연산자의 우선순위(높은 순)
  - 괄호 (( ))
  - 거듭제곱(^, **)
  - 곱하기(*),
  - 나누기(/)
  - 더하기(+),
  - 빼기(-)

- 동일한 우선순위의 산술 연산자가 나열된 경우 왼쪽이 오른쪽보다 우선순위를 갖게

산술 연산자(Arithmetic Operator)

연산자	설명	입력내용	결과내용
+	더하기	3 + 4	7
-	빼기	3 – 4	-1
*	곱하기	3 * 4	12
/	나누기	3 / 4	0.75
**	거듭제곱	3 ** 4	81
^	거듭제공	3 ^ 4	81
%/%	몫	13 %/% 4	3
%%	나머지	13 %% 4	1

할당 연산자(Allocation Operator)

- 어떤 객체의 이름(변수 이름, 데이터 이름)에 특정한 값을 저장할 때 사용하는 연산자
- 객체에 무엇이 있는지를 일부러 새로운 명령어를 실행하지 않아도 알 수 있는 기능

연산자	설명	입력내용	결과 내용
<-	오른쪽의 값을 왼쪽의 이름에 저장	x <-3	• R Studio왼족 하단에 x, y, z라는 변수 생성 • 각각의 값이 3, 4, 5가 정의됨
=	오른쪽의 값을 왼쪽의 이름에 저장	y = 4	
->	왼쪽의 값을 오른쪽의 이름에 저장	5 -> z	

비교 연산자(Relational Operator)

- 두 개 값에 대한 비교로써 맞으면 TRUE, 맞지 않으면 FALSE를 반환하여
  - 비교 연산의 최종적인 결과가 갖는 값
- 비교 연산자의 종류
  - 크다
  - 크거나
  - 같다
  - 작다
  - 작거나 같다
  - 같다
  - 같지 않다
  - 아니다

비교 연산자(Relational Operator)

연산자	설명	입력내용	결과내용
>	크다	3 > 4	FALSE
>=	크거나 같다	3 >= 4	FALSE
<	작다	3 < 4	TRUE
<=	작거나 같다	3 <= 4	TRUE
==	같다	3 == 4	FALSE
!=	같지 않다	3 != 4	TRUE
!	부정	!(3 == 4)	TRUE

- 연산자 안에 공백이 있으면 에러(Error)가 발생하는 점 체크
- ('크거나 같다'를 사용할 때에 '>공백='로 사용하면 안 됨)

논리 연산자(Logical Operator)

- 두 개 이상의 조건을 비교하여 결과를 냄
- &, &&는 모든 조건이 참일 때에만 최종적인 결과가 TRUE가 됨
- |, ||는 조건 중에서 하나라도 참이면 최종적인 결과가 TRUE가
- 벡터(Vector)가 오면 &와 &&의 결과 또는 |와 ||의 결과에는 차이가 있음

연산자	설명	입력 내용
&	AND	(조건1) & (조건2)
&&	AND	(조건1) && (조건2)
|	OR	(조건1) | (조건2)
||	OR	(조건1) || (조건2)

논리 연산자(Logical Operator)

- &
  - AND의 개념 두 개(또는 그 이상)의 조건을 동시에 만족할 때만 TRUE가 되는 논리 연산
  - &는 데이터가 하나인 경우나 데이터가 두 개 이상인 경우

- &&
  - AND의 개념으로 두개(또는 그 이상)의 조건을 동시에 만족할 때만 TRUE가 되는 논리 연산
  - 데이터가 하나인 경우에만 가능
  - 벡터(Vector)인 경우 벡터(Vector)의 첫 번째만 작동하고 나머지는 작동하지 않음

논리 연산자(Logical Operator)

- |
  - OR의 개념
  - 두 개(또는 그 이상)의 조건 중에서 하나만 만족하여도 TRUE가 되는 논리 연산
  - 데이터가 하나인 경우나 두 개 이상인 경우

- ||
  - OR의 개념으로 두 개(또는 그 이상)의 조건 중에서 하나만 만족하여도 TRUE가 되는 논리 연산
  - 데이터가 하나인 경우에만 가능
  - 벡터(Vector)인 경우에는 벡터(Vector)의 첫 번째만 작동하고 나머지는 작동하지 않음

R의 데이터 유형

- 기본 데이터 유형
- 특수 형태의 데이터 유형
- 데이터 유형 알아내기
- 데이터 유형의 우선순위
- 데이터 유형의 강제변환

기본 데이터 유형

- R에서 데이터의 기본적인 속성으로 데이터의 유형이 있음
- 데이터가 어떤 값(숫자, 문자)으로 이루어져 있는가를 의미
- 데이터 유형에는 기본적인 것과 특수한 형태로 구성

기본 데이터 유형

- 기본적인 데이터 유형에는 수치형, 문자형, 논리형, 복소수형이 있음
- 수치형, 문자형, 논리형이 자주 사용
- 복소수형은 수학분야를 다룰 때에 사용

- 수치형(Numeric) : 숫자로 되어 있으며, 정수형(Integer)과 실수형(Double)이 있음
- 논리형 (Logical) : 참과 거짓의 논리값으로 TRUE(or T)나 FALSE(or F) 를 가짐
- 문자형(Character) : 하나의 문자 또는 문자열로 되어 있으며, “” 또는 ‘’로 묶여 있음
- 복소수형(Complex) : 실수와 허수로 이루어진 복소수

특수 형태의 데이터 유형

- NULL
  - 존재하지 않는 객체로 지정할 때 사용
- NA
  - Not Available의 약자로 결측치(Missing value)를 의미
- NaN
  - Not available Number의 약자로 수학적으로 계산이 불가능한 수를 의미
  - 예 : sqrt(-3)로 음수에 대한 제곱근은 구할 수 없음
- Inf
  - Infinite의 약자로 양의 무한대
- -Inf
  - 음의 무한대

데이터 유형 알아내기

- 데이터의 유형을 알려주는 함수
  - mode() 함수
  - is로 시작하는 함수

mode() 함수

- 문자형 형태로 최종적인 결과를 알려줌 (“numeric”, “character”, “logical”, “complex”)중에 하나로 표현

- x1 = 3
- x2 = “Hello R”
- x3 = FALSE
- x4 = 3 – 2i

- x1(수치형)의 데이터 유형 – “numeric”
- x2(문자형)의 데이터 유형 - “character”
- x3(논리형)의 데이터 유형 - “logical”
- x4(복소수형)의 데이터 유형 - “complex

is로 시작하는 함수

- is로 시작하는 함수들의 최종적인 결과는 TRUE 또는 FALSE 형태로 나타남

함수명	설명	입력내용	결과내용
is.numeric()	수치형 여부	is.numeric(데이터)	TRUE or FALSE
is.integer()	정수형 여부	is.integer(데이터)	TRUE or FALSE
is.double()	실수형 여부	is.double(데이터)	TRUE or FALSE
is.character()	문자형 여부	is.character(데이터)	TRUE or FALSE
is.logical()	논리형 여부	is.logical(데이터)	TRUE or FALSE
is.complex()	복소수형 여부	is.complex(데이터)	TRUE or FALSE
is.null()	NULL 여부	is.null(데이터)	TRUE or FALSE
is.na()	NA 여부	is.na(데이터)	TRUE or FALSE
is.finite()	유한 수치 여부	is.finite(데이터)	TRUE or FALSE
is.infinite()	무한 수치 여부	is.infinite(데이터)	TRUE or FALSE

데이터 유형의 우선순위

- 대표적인 데이터 유형(수치형, 문자형, 논리형, 복소수형)에는 우선순위가 있음
- 벡터와 같은 데이터에서 발생
  - 벡터는 하나의 유형만 가질 수 있음
  - 벡터를 만들 때에 여러 가지의 유형을 넣어도 최종적인 결과에는 하나의 유형으로 변경됨
- 4가지 데이터 유형의 우선순위
  - 문자형(Character)
  - 복소수형(Complex)
  - 수치형(Numeric)
  - 논리형(Logical)

    - x1 = c(1, “Hello”, TRUE, 2+3i)
    - [1] “1” “Hello” “TRUE” “2+3i”
    - x1은 네 개의 값을 가지는 벡터, 네 가지 유형으로 입력
    - x1이라는 벡터가 가지는 최종적인 유형은 한 가지, 문자형임
    - 데이터 이름을 주고 실행하면 데이터가 가지는 값을 보여줌

데이터 유형의 강제 변경하는 함수

연산자	설명	입력내용	결과내용
as.numeric()	수치형으로 변환	as.numeric(데이터)	변환되거나NA
as.interger()	정수형으로 변환	as.interger(데이터)	변환되거나NA
as.double()	실수형으로 변환	as.double(데이터)	변환되거나NA
as.character()	문자형으로 변환	as.character(데이터)	변환되거나NA
as.logical()	논리형으로 변환	as.logical(데이터)	변환되거나NA
as.complex()	복소수형으로 변환	as.complex(데이터)	변환되거나NA

- 데이터 유형 중 우선순위가 낮은 형태에서 우선순위가 높은 형태로는 강제적으로 유형 변환이 가능함
- 우선순위가 높은 형태에서 우선순위가 낮은 형태로의 변환은 일부만 가능하기 때문에 경우에 따라서 강제적으로 유형이 변경되지 않을 수 있음

벡터란 무엇인가?

- 벡터가 가지는 하나의 값 = 원소(Element)
- 원소들이 어떤 데이터 유형으로 이루어졌는지, 원소의 개수는 몇 개인지, 원소의 이름은 어떻게 되어 있는지 를 나타내는 것을 벡터의 속성이라고 함

벡터의 데이터 유형 확인하기

- 벡터가 가지는 값의 데이터 유형을 알아보기 위해 mode(), is.numeric(), is.character(), is.logical(), is.complex() 함수를 사용

- 함수의 사용방
  - mode(벡터)
  - is.numeric(벡터)
  - is.character(벡터)
  - is.logical(벡터)
  - is.complex(벡터)

벡터의 데이터 유형 확인하기

- 데이터 유형을 알아보는 방법
  - 27, 35, 47, 41의 4개의 수치형 값으로 이루어진 v1 벡터를 생성, v1의 데이터 유형 확인하기

  - v1 = c(27, 35, 47, 41)
  - mode(v1) => mode() 함수를 쓰면 v1의 데이터 유형이 수치형이라고 알려줌
  - is.numeric(v1) => is.numeric() 함수를 쓰면 TRUE라는 결과를 알려줌

원소의 개수

- 벡터가 몇 개의 원소(element)를 가지고 있는지를 원소의 개수(또는 데이터의 개수)라고 함
- length() 함수를 사용하여 확인 가능함
  - length(벡터) -> length() 함수를 쓰면 v1 벡터의 원소는 개수는 4개라고 알려줌
  - length

원소의 이름

- 기본적으로는 벡터의 원소에 대한 이름은 없으나 이름을 부여할 수 있음
- names() 함수를 사용
  - 원소의 이름이 무엇인지 알 수 있음
  - 원소의 이름을 새롭게 부여할 수 있음
- names() 함수 사용방법(이름 확인하기)

  - names(벡터) => v1에 대한 원소의 이름이 없기 때문에 NULL이라는 결과를 알려줌
  - names(v1) => NULL은 객체(object)가 존재하지 않는다는 뜻

원소의 이름

- names() 함수 사용방법(이름 부여하기)
  - names(v1) = c(“Kim”, “Lee”, “Park”, “Choi”) => v1에 대한 원소의 이름으로 “Kim”, “Lee”, “Park”, “Choi”를 부여
  - names(v1) => 원소의 이름을 부여한 후에 names() 함수를 다시 실행하면 v1이 가지는 각각의 원소에 대한 이름이 출력됨

벡터의 인덱싱

- 벡터가 가지는 원소들 중에서 일부의 원소를 추출할 때는 대괄호([])를 사용
- 대괄호 안에 추출하기 원하는 원소의 위치(index)를 수치로 입력
- R에서는 벡터의 첫 번째 원소에 대한 위치는 1부터 시작
- 두 개 이상의 원소를 추출할 때에는 c(), :, seq(), sequence() 등을 사용
  - 다른 컴퓨터 프로그램 언어는 첫 번째 원소에 대한 위치를 0으로 인식하는 경우가 있기 때문에 차이점을 잘 인지해야 함

벡터의 인덱싱

- 벡터에서 일부의 원소를 추출하는 방법
- 벡터[index]
  - 5명의 몸무게(57, 81, 65, 49, 72)를 저장하는 weight라는 변수를 생성

  - weight = c(57, 81, 65, 49, 72)
  - weight[1] => 첫 번째 데이터인 57을 가져옴
  - weight[2] => 두 번째 데이터인 81을 가져옴
  - weight[c(1, 4, 5)] => 1, 4, 5번째 있는 데이터인 57, 49, 72를 가져옴
  - weight[2:4] => 2, 3, 4번째에 있는 데이터인 81, 65, 49를 가져옴
  - weight[-c(1, 4, 5)] => 1, 4, 5번째 있는 데이터를 제외하고 나머지 데이터를 가져오라는 뜻 즉 2, 3번째 위치에 있는 데이터인 81, 65를 가져옴

  - 가져오고 싶은 데이터가 연달아 있지 않고, 특정한 규칙이 없다면 c() 함수를 이용하여 가져오고 싶은 위치를 지정
  - 가져오는 데이터가 연달아 있는 경우에는 :(콜론)을 사용하면 편리

벡터의 연산

- 수치형 벡터로 된 두 개 이상의 벡터 간에 사칙연산 가능
- R의 강점은 벡터의 연산이 편리하고 빠르다는 점임

벡터의 길이가 동일한 경우

- 연산에 사용되는 벡터들의 원소의 개수가 동일한 경우

  → 벡터들 간의 사칙연산이 가능하며 최종적인 결과는 벡터가 됨
- 벡터들 간의 연산이 될 때에는 각 벡터에 있는 동일한 위치의 값들 간이 연산이 됨
  - 예 : 수치형 벡터 v1, v2가 있을 때

    v1+v2의 연산을 하면 v1[1]+v2[1]의 연산이 됨

벡터의 길이가 동일한 경우

- v1에는 1, 2, 3을 가지고, v2는 4, 5, 6을 갖도록 생성한 후에 연산
  - v1 = 1:3
  - v2 = 4:6
  - v1 + v2
  - v1 – v2
  - v1 * v2
  - v1 / v2
  - v1 ** v2

벡터의 길이가 동일하지 않는 경우

- 두 벡터가 가지는 원소의 개수가 다르더라도 결과적으로 연산이 됨
- 벡터 자체는 변하지 않지만 연산과정에서 원소의 개수가 적은 쪽의 벡터는 원소 개수가 많은 쪽의 벡터와 동일하게 원소의 개수를 맞춤
  - 원소 개수가 차이가 나는 만큼 임시적으로 데이터가 생성됨

- 재사용 규칙(reycling rule)
  - v1은 3개의 데이터, v2는 6개의 데이터가 있다면 v1과 v2 벡터 간의 연산을 했을 경우
    - v1의 데이터는 추가적으로 3개가 더 생성되어 6개가 됨
    - 새롭게 생성되는 3개의 데이터는 v1이 가지고 있는 값을 순서적으로 새롭게 생성되는 데이터에 지정
  - 재사용 규칙은 R이 가지는 중요한 특징

벡터의 길이가 동일하지 않는 경우

- v1에 1, 2, 3을, v2에는 1, 2, 3, 4, 5, 6을 저장하고 있을 때, 두 벡터 간의 연산
  - v1 = 1:3 => v1은 3개의 수치형 데이터를 가지는 벡터
  - V2 = 1:6 => v2은 6개의 수치형 데이터를 가지는 벡터
  - V1 + v2 => v1는 변하지 않지만 연산과정에서만 추가적으로 3개의 데이터를 가짐
  - 새롭게 생성되는 4, 5, 6번째의 데이터는 v1의 1, 2, 3번째에 있는 값을 순서대로 갖게 됨 => 1, 2, 3, 1, 2, 3
    - 연산과정에서 데이터의 개수가 동일하게 되고, 동일한 위치 간의 연산이 가능하게 됨 => v1[1] + v2[1], v1[2] + v2[2], ..., v1[6] + v2[6]
    - 즉, 1 + 10, 2 + 11, 3 + 12, 1 + 13, 2 + 14, 3 + 15
    - 결과값 11, 13, 15, 14, 16, 18

벡터의 길이가 동일하지 않는 경우

- v1은 1, 2, 3을 가지는 벡터이고, v2는 1, 2, 3, 4, 5, 6, 7, 8, 9, 10을 가지는 벡터일 때
  - v1 = 1:3
  - v2 = 1:10
  - v1 + v2

  - 재사용규칙이 동일하게 적용되어 v1은 연산과정에서 7개가 추가적으로 생성 추가적으로 생성된 데이터의 값은 v1에 있는 1, 2, 3을 순서적으로 가짐

    => 1, 2, 3, 1, 2, 3, 1, 2, 3, 1
  - v1과 v2 벡터 간에 연산, 경고 메시지 발생(warning message)
    - v1의 데이터 개수와 v2의 데이터 개수 간에 배수가 존재하지 않기 때문임(3과 10)

  - R은 두 벡터의 데이터 개수가 배수의 관계가 아닐 때 사용자가 원하는 결과가 아닐지도 모르니 잘 살펴보라는 의미에서 경고 메시지를 발생시킴

하나의 값으로 이루어진 벡터

- s1 = 10
- s2 = “Hello”
- s3 = FALSE
- s4 =1 – 3i
  - s1는 수치형, s2는 문자형, s3은 논리형, s4는 복소수형으로 된 벡터임
  - 데이터가 한 개이기 때문에 벡터가 갖는 데이터의 유형은 오직 하나임

두개 이상의 값으로 이루어진 벡터

- 두 개 이상의 값으로 이루어진 벡터를 생성할 때와 하나의 값으로 이루어진 벡터를 생성하는 방법이 다름
- 두 개 이상의 값으로 이루어진 벡터를 생성하는 것이 대표적인 방법 c(), :, seq(), sequence(), rep() 소개

두개 이상의 값으로 이루어진 벡터

- c()
  - c() 함수는 combine 또는 concatenate의 약자
  - 벡터를 생성하는 가장 대표적인 방법
  - 네 가지 유형(수치형, 문자형, 논리형, 복소수형)에 적용 가능
    - 수치형 벡터, 문자형 벡터, 논리형 벡터, 복소수형 벡터 생성 가능
  - 규칙이 없는 데이터로 이루어진 벡터를 생성할 때 사용
  - 규칙이 있을 때에는 다른 기능을 이용

두개 이상의 값으로 이루어진 벡터

- c() 함수를 사용하는 방법

Argument	**설명**
…	• 기본적으로는 하나의 값을 콤마로 구분하여 입력하면 두 개 이상의 값을 가지는 벡터가 됨 • 또한 두 개 이상의 값으로 이루어진 벡터를 콤마로 넣어도 최종적으로는 하나의 벡터가 되도록 해줌

- 수치형 벡터인 v1, 문자형 벡터인 v2, 논리형 벡터인 v3을 생성하면 다음과 같음
  - v1 = c(3, 10, 12)
  - v2 = c("a", "p", "p", "l", "e")
  - v3 = c(TRUE, FALSE, TRUE, FALSE)
    - v1은 3개의 수치형으로 이루어진 벡터
    - v2는 3개의 문자형으로 이루어진 벡터
    - v3은 4개의 논리형으로 만들어진 벡터
  - a처럼 큰 따옴표를 생각하고 "FALSE"에 큰 따옴표를 사용하는 것에 주의!
  - 문자형은 큰 따옴표나 작은 따옴표로 감싸야 하고, 논리형은 감싸지 않음

두개 이상의 값으로 이루어진 벡

- c() 함수는 벡터들을 하나로 합쳐서 하나의 새로운 벡터를 생성할 수도 있음
- 새로운 v1 벡터와 v2를 생성, c() 함수를 이용하고, v1, v2 벡터를 연결하여 v3이라는 새로운 벡터를 생성할 수 있음
  - v1 = c(1, 3, 5)
  - v2 = c(10, 30, 50)
  - v3 = c(v1, v2)
    - v3의 새로운 벡터 생성
  - c() 함수는 다른 데이터 형태에서 일부의 데이터를 추출할 때에도 사용
  - 규칙이 없는 특정한 행이나 열의 데이터를 추출할 때 사용

두개 이상의 값으로 이루어진 벡터

- :
  - 콜론(:)은 수치형에만 적용
  - 1씩 증가되거나 1씩 감소되는 규칙이 있는 값으로 이루어진 벡터를 생성할 때 사용
  - start:end 구조로 사용
  - start, end는 숫자 start > end이면 1씩 감소 start < end이면 1씩 증가 start=end이면 start 또는 end가 됨
  - 시작은 무조건 start에서 시작, end를 넘지 않음
    - v1 = 1:5
      - V1 = 1:5
      - start는 1, end는 5
      - start < end가 성립하므로 1씩 증가
      - start부터 시작해서 end를 넘지 않을 때까지 1씩 증가하는 수치형으로 이루어지는 벡터 =>1부터 시작해서 5를 넘지 않을 때까지 1씩 증가하는 수치형 벡터가 생성
  - 콜론(:)도 벡터를 생성할 때 사용하지만, 다른 데이터 형태에서 연달아 있는 특정한 행이나 열의 데이터를 추출할 때 사용할 수 있음

두개 이상의 값으로 이루어진 벡터

- seq() 함수는 sequence의 약자
- 콜론(:)의 확장 또는 일반화
- 콜론(:)은 1씩 증가하거나 1씩 감소하는 수치형으로 이루어진 벡터를 생성하지만, seq() 함수는 1 이외의 증가 또는 감소가 되는 규칙 있는 수치형 벡터를 생성함

두개 이상의 값으로 이루어진 벡터

- seq() 함수를 사용하는 방법

argument	**설명**
from	시작값
to	끝값
by	얼마씩 증가 또는 감소시킬지를 정하는 단계값 단, 감소시킬 경우에는 부호가 음수이어야 하며 그렇지 않으면 에러(error)가 발생함

- v1 = seq(from=1, to=5, by=1)
  - v1=seq(from=1, to=5, by=1) 1부터 시작해서 5를 넘지 않을 때까지 1씩 증가하는 수치형 벡터가 생성 => v1은 1, 2, 3, 4, 5로 됨

- seq() 함수에는 from, to, by라는 argument가 있음
- R을 사용하면서 함수에 있는 argument는 가능하면 생략하지 않고 쓰도록 함 (함수 안에 있는 값들이 무엇인지 본인과 코드를 보는 다른 사람에게도움이 됨)
- argument의 순서는 문법에 영향을 주지 않지만 최대한 순서를 중요하게 생각해서 작성하는 것을 추천

두개 이상의 값으로 이루어진 벡터

- sequence()
  - sequence() 함수는 sequence(숫자) 형태로 사용
  - 1과 지정한 '숫자' 사이의 정수로 이루어진 수치형 벡터를 생성

argument	**설명**
nvec	수치형인 벡터를 지정한다

  - v1 = sequence(10)
  - V1 => v1=sequence(10) 1~10 사이의 정수 형태인 수치형 벡터가 생성 v1은 1, 2, 3, 4, 5, 6, 7, 8, 9, 10이 됨

두개 이상의 값으로 이루어진 벡터

- rep()
  - rep() 함수는 replicate의 약자(복사하다, 반복하다의 뜻)
  - rep() 함수는 수치형, 문자형, 논리형, 복수형으로 된 벡터를 생성할 수 있음
  - rep() 함수에 지정된 데이터를 복사해 주는 기능

- rep() 함수를 사용하는 방법

argument	**설명**
x	복사하고 싶은 벡터를 지정
times	times에 지정된 벡터가 하나의 수치라면, 복사하고 싶은 벡터 전체를 times만큼 복사하여 새로운 벡터를 생성함
	times에 지정된 것이 수치형 벡터이면, 복사하고 싶은 벡터의 원소 각각을 times에 지정된 벡터의 원소만큼 복사해서 새로운 벡터를 생성함
each	each에 지정된 숫자만큼 스칼라나 벡터의 원소를 각각 복사해서 하나의 벡터가 생성하도록 함 ※여기에 오는 수는 양수이다.
length.out	생성되는 벡터의 길이를 지정하며, 양수이어야 함

두개 이상의 값으로 이루어진 벡터

- rep()
  - v1 = rep("a", times=5)
    - v1 = rep("a", times=5) "a"라는 문자형 데이터를 5번 복사해서 문자형 벡터인 v1을 생성