R컴퓨팅

8강 프로그래밍 구조 II

정보통계학과 장영재 교수

1 기본적인 R 함수

2 프로그래밍의 기본 구조

1 기본적인 R 함수

1 기본적인 R 함수

1 수치적 함수

함수	예
• pi	> pi [1] 3.141593 > options(digits=20) > pi [1] 3.141592653589793

함수	예
• sin(x): sin 함수	> sin(10) [1] -0.54
• cos(x): cosine 함수	> cos(10) [1] -0.84
• tan(x): tangent 함수	> tan(10) [1] 0.65
• asin(x) : arcsin 함수	> asin(1) [1] 1.6
• acos(x): arc cosine 함수	> acos(0) [1] 1.6
• atan(x):arc tangent 함수	> atan(0.6) [1] 0.54

함수	예	
• log(x) : 자연로그 함수	예1) > log(2) [1] 0.7	예2) > x<-3 > y<-4 > log(x+y) [1] 1.9
• log10(x): 상용 로그 함수	예1) > log10(10) [1] 1	예2) > x<-3 > y<-14 > log(x+y) [1] 1.2
• exp(x): 지수 로그 함수	> exp(10) [1] 22026	
• sqrt(x) : 루트함수	> sqrt(8) [1] 2.8	

1 기본적인 R 함수

1 수치적 함수

함수	예
• min(x) :벡터에서 최소값	> x<-c(1,2,-3,4) > min(x) [1] -3
• max(x): 벡터에서 최대값	> x<-c(1,2,-3,4) > max(x) [1] 4
• min(x1, x2,...) : 전체 벡터 원소 중에서 최소값	> x1<-c(1,2,-3,4) > x2<-c(2,4,-6,7) > min(x1,x2) [1] -6
• range(x): 벡터의 범위 -> c(min(x), max(x))	> x<-c(1,2,-3,4) > range(x) [1] -3 4 > c(min(x), man(x)) [1] -3 4

함수	예
• pmin(x1,x2) : 두 벡터의 상응하는 원소들 중 작은 값	> x1<-c(1,2,-3,4) > x2<-c(2,4,-6,7) > pmin(x1,x2) [1] 1 2 -6 4
• pmax(x1,x2) : 두 벡터의 상응하는 원소들 중 큰 값	> x1<-c(1,2,-3,4) > x2<-c(2,4,-6,7) > pmin(x1,x2) [1] 2 4 -3 7

1 기본적인 R 함수

1 수치적 함수

- R의 장점 중 한 가지는 다양한 통계함수가 내장되어 있어서 기본적인 통계량 산출이 용이하다는 것

함수	예
• mean(x1): 평균	> x1<-c(1,2,3,4,5,6) > mean(x1) [1] 3.5
• sd(x1): 표준 편차	> x1<-c(1,2,3,4,5,6) > sd(x1) [1] 1.9
• var(x1): 분산	> x1<-c(1,3,6,9,12,3,2) > var(x1) [1] 16
• median(x1): 중앙값(중위수)	> x1<-c(1,3,6,9,12,3,2) > median(x1) [1] 3

함수	예
• quantile(x,p): (100*p)%에 해당하는 값	> x1<-c(1,2,3,4,5,6,7,8,9,10) > quantile(x1, 0.5) [1] 50% [1] 5.5
• cor(x,y) : 상관 계수	> x<-c(1,2,3,4,5,6,7,8,9,10) > y<-c(10,9,8,7,6,5,4,3,2,5) > cor(x,y) [1] -0.91

2 프로그래밍의 기본 구조

2 프로그래밍의 기본 구조

1 조건문

- 조건문이라는 것은 특정한 조건을 만족했을 경우에만 프로그램 코드를 수행하는 제어 구문을 의미
- 조건문에는 항상 논리 연산이 수반되며 조건문의 구체적인 표현식은 조건의 개수, 조건문의 위치, 조건에 따른 명령수행 방식 등에 따라 구분

[1] if(조건) 실행문

if 조건문은 괄호 안의 조건이 참이면 실행문을 수행하는 표현식으로 실행문이 여러 개일 경우 실행문을 대괄호({ })로 묶어서 사용

2 프로그래밍의 기본 구조

1 조건문

보기 8-1 : 일정한 기준을 만족할 때에만 명령을 수행하는 if 조건문

```
> x <- c(1,2,3,4)
> y <- c(2,1,4,5)
> if(sum(x) < sum(y)) print(x)    # x의 합이 y의 합보다 작을 때 x를 출력
[1] 1 2 3 4
> if(sum(x) < sum(y) {
+ print(mean(x))
+ print(var(x))
+ }              # x의 합이 y의 합보다 작으면 x의 평균과 분산을 출력
[1] 2.5
[1] 1.66667
```

2 프로그래밍의 기본 구조

1 조건문

[2] if(조건) else 조건 실행문

if (조건) else 조건 실행문은 괄호 안의 조건이 참이면 참일 때의 실행문을 수행하고 거짓일 때는 참이 아닐 때의 실행문을 수행하는 표현식

2 프로그래밍의 기본 구조

1 조건문

- 보기 8-2 : 보기 4-4에서 사용한 동일한 벡터 x, y를 정의하고, 평균값을 비교하여 ”Mean(x) > Mean(y)" 또는 “Mean(x) < Mean(y)" 등으로 출력하는 조건문

```
> x <- c(1,2,3,4)
> y <- c(2,1,4,5)
> if(mean(x)>mean(y)) print("Mean(x)>Mean(y)") else
+ print("Mean(x)<Mean(y)")
[1] "Mean(x)<Mean(y)"
> #x의 평균이 y의 평균보다 크면 "Mean(x)>Mean(y)" 출력,
> #x의 평균이 y의 평균보다 작으면 "Mean(x)<Mean(y)" 출력
```

2 프로그래밍의 기본 구조

1 조건문

```
> if(mean(x)>mean(y)) {       # x의 평균이 y의 평균보다 크면
+ print(mean(x))
+ print(var(x))          # x의 평균과 분산을 출력
+ } else {           # 그렇지 않으면
+ print(mean(y))
+ print(var(y))          # y의 평균과 분산을 출력
+ }              # if 와 else의 조건문들이 대괄호로 묶여 있음
[1] 3
[1] 3.3333
```

2 프로그래밍의 기본 구조

1 조건문

[3] 중첩 조건문

중첩 조건문은 조건문 내에 조건문이 위치하는 형태

- 보기 8-3 : ① 보기 4-4에서 사용한 동일한 벡터 x, y를 정의하고, x의 길이가 5라는 조건하에 x의 합이 10이면 “length=5, sum=10"을 출력하고 x의 길이가 5가 아니면 ”length=4, sum=10"을 출력 ② x의 길이가 4라는 조건하에 x의 합이 10이면 "length=4, sum=10"을 출력하고 10이 아니면 "length=5, sum=10"을 출력

2 프로그래밍의 기본 구조

1 조건문

```
> x <- c(1,2,3,4)
> y <- c(2,1,4,5)
> if(length(x)==5) {          # 외부 조건문
+ if(sum(x) == 10) print("length=5, sum=10")          # 내부 조건문
+ } else {
+ print("length=4, sum=10")
+ }
[1] "length=4, sum=10"
> if(length(x)==4) {                  # 외부 조건문
+ if(sum(x) == 10) print("length=4, sum=10") else     # 내부 조건문
+ print("length=5, sum=10")
+ }
[1] "length=4, sum=10"
```

2 프로그래밍의 기본 구조

1 조건문

[4] ifelse 조건문

ifelse 조건문은 조건문의 괄호 내에 조건, 조건이 참일 때의 실행문, 조건이 참이 아닐 때의 실행문이 모두 위치

- 보기 8-4 : 보기 4-4에서 사용한 동일한 벡터 x, y를 정의하고, x가 y보다 작으면 x의 값을 반환하고 그렇지 않으면 y값을 반환하며 x-y의 합이 0보다 크면 “positive"을 출력하고 x-y의 합이 0보다 작다면 "negative"을 출력하고 0보다 크지도 작지도 않으면 ”zero"를 출력

2 프로그래밍의 기본 구조

1 조건문

```
> x <- c(1,2,3,4)
> y <- c(2,1,4,5)
> ifelse(x<y, x, y)
[1] 1 1 3 4

> ifelse(sum(x-y) > 0, "positive", ifelse(sum(x-y) < 0 , "negative",
"zero"))
[1] "negative"
```

2 프로그래밍의 기본 구조

1 조건문

[5] switch문

switch문은 switch(매개변수, 실행문1, 실행문2, ...)와 같이 사용하며, 매개변수의 값에 따라 조건에 맞는 실행문을 찾아 수행

› 보기 8-5: switch의 매개변수 값이 문자열을 갖는 경우

```
〉 x 〈– c(1,2,3,4)
〉 type〈–"var"
〉 switch(type, mean=mean(x), median=median(x), sum=sum(x),
var=var(x))
[1] 1.66667
```

2 프로그래밍의 기본 구조

1 조건문

› 보기 8-6: 다음은 switch의 매개변수 값이 정수값을 갖는 경우

```
〉 x 〈– c(1,2,3,4)
〉 switch(1, mean(x), sum(x), var(x))
[1] 2.5
```

switch문에서 매개변수 값이 정수 값을 갖게 되면
이 정수값에 해당하는 순서의 실행문을 수행

2 프로그래밍의 기본 구조

2 반복문

[1] for 반복문

for 반복문은 for(변수 in 반복횟수) {실행문}과 같은 형태로서
대괄호 내에 위치한 실행문을 반복횟수만큼 실행하는 구문

› 보기 8-7: for 반복문을 실행하여 아래와 같은 결과를 출력해 보기

```
[1] 1
[1] 2 2
[1] 3 3 3
[1] 4 4 4 4
[1] 5 5 5 5 5
> for(i in 1:5) print(rep(i,i))
```

2 프로그래밍의 기본 구조

2 반복문

보기 8-8: for 반복문을 실행하여 1에서 10까지 합을 구하기

```
> x<-0
> for(i in 1:10)
+ x <- x + i
> x
[1] 55
```

2 프로그래밍의 기본 구조

2 반복문

[2] while 반복문

while 반복문은 while(조건) {실행문}의 형태로 쓰이며 조건이 참일 때까지 실행문을 수행하는 반복문

› 보기 8-9: while 반복문을 실행하여 보기 4-9에서 제시된 것과 동일한 결과를 출력

```
> i<-1
> while(i<=5) {
+ print(rep(i,i))
+ i<-i+1    # 1씩 증가시켜 자신에게 할당
+ }
```

2 프로그래밍의 기본 구조

2 반복문

› 보기 8-10: while 반복문을 실행하여 1에서 10까지 합을 구하기

```
> i<-1
> x<-0
> while(i <= 10) {
+ x<-x + i
+ i<-i + 1
+ }
> x
1] 55
```

2 프로그래밍의 기본 구조

2 반복문

[3] repeat 반복문

repeat 반복문은 repeat {실행문}과 같은 형태로 사용되며 실행문을 계속 수행하다가 break를 만나면 반복수행을 멈추고 작업을 종료

› 보기 8-11: repeat 반복문을 실행하여 보기 4-9에서 얻었던 결과를 똑같이 출력하기

```
> i<-1
> repeat {
+ if(i > 5) break
+ print(rep(i,i))
+ i<-i+1
+ }
```

2 프로그래밍의 기본 구조

2 반복문

› 보기 8-12: repeat 반복문을 실행하여 1에서 10까지 합을 구하기

```
> i<-1
> x<-0
> repeat {
+ if(i > 10) break
+ x<-x + i
+ i<-i + 1
+ }
> x
[1] 55
```

2 프로그래밍의 기본 구조

3 무조건 분기문

[1] break 분기문

break 분기문은 루프를 탈출하는 제어문

- 중첩 루프가 사용되었을 경우, break가 위치한 해당 루프에서만 빠져 나오게 된다는 점에 유의

- 보기 8-13: 1에서 10까지 합을 구하는 루프에서 합이 25이하가 되는 시점까지 각 단계별 합계를 출력

2 프로그래밍의 기본 구조

3 무조건 분기문

```
> x<-0
> for(i in 1:10) {
+ x <- x + i
+ if(x > 25) break # 만약 각 단계까지의 합계 x가 25보다 크다면
+ # break문을 수행하여 반복문을 벗어나게 됨
+ print(x)
+ }
[1] 1
[1] 3
[1] 6
[1] 10
[1] 15
[1] 21
```

2 프로그래밍의 기본 구조

3 무조건 분기문

› 보기 8-14: for 반복문을 중첩 반복문 형태로 사용하여 아래와 같은 결과를 출력해 보기

```
[1] 1 * 1 = 1
[1] 2 * 1 = 2
[1] 3 * 1 = 3
> for(i in 1:9) {
+ if(i > 3) break
+ for(j in 1:9) {
+ if(j > 1) break
+ cat(i, "*", j, "=", i*j, " ₩n")
+ }
+ }
```

2 프로그래밍의 기본 구조

3 무조건 분기문

[2] next 분기문

next는 break보다 더 강제적인 제어문

- 반복문을 수행하더라도 조건을 충족하면 next 이후의 실행문을 수행하지 않고 계속 건너뛰게 됨

- 보기 8-15: while 반복문과 next 분기문을 실행하여 1에서 10까지 변수값을 증가시키되 8보다 작을 때까지는 값을 출력하지 않고 8이상인 값과 이들의 합을 출력

2 프로그래밍의 기본 구조

3 무조건 분기문

```
> i<-1
> x<-0
> while(i < 10) {
+ i<-i+1
+ if(i<8) next   # next 이후의 명령은 if문의 조건이 충족되는 한 미수행
+ print(i)
+ x<-x + i
+ }
[1] 8
[1] 9
[1] 10
```

R컴퓨팅