Note 8.Function & Control

Introduction to Statistical Programming

Chanmin Kim

Department of Statistics
Sungkyunkwan University

2022 Spring

Function

- Functions: *function name* = function(*arguments*){ }.
  - Built-in functions.
  - User-define functions.

```
> avg <- function(x)
+ {
+   mean <- sum(x) / length(x)
+   return(mean)
+ }

> avg <- function(x) sum(x) / length(x)

> x <- c(3,6,2,7,7)
> avg(x)
[1] 5
```

Loops

- `for (`*i* `in` *x*`) { }`:
  - Iterate statements in `{ }`.
  - 1st iteration: `i = x[1]`, 2nd iteration: `i=x[2]`,...
  - Known # of repetition = length of *x*.
  - When we want to use elements of *x* in statements.
  - `break`: Early termination.
  - `next`: Go to the next iteration (skip the current iteration).

- `while(`*condition*`) { }`:
  - Iterate statements in `{ }`, while *condition* is `TRUE`.
  - `break`: Early termination is possible, even if *condition* is `TRUE`.

- `repeat { }`:
  - Similar to `while`, but there is no *condition* for termination.
  - For termination, `break` is used.

Loops

```
> x <- c(5,6,10,7,12); z <- NULL
> for (i in x) z <- c(z,i+3)
> z
[1]  8  9 13 10 15

> x <- c(5,6,10,7,12); z <- NULL
> for (i in x)
+ {
+   z <- c(z,i+3)
+   if (i >= 10) break
+ }
> z
[1]  8  9 13

> x <- c(5,6,10,7,12); z <- NULL
> for (i in x) {if (i >= 10) next;  z <- c(z,i+3)}
> z
[1]  8  9 10
```

Loops

```
> x <- 0
> while  (x < 5) x <- x + 1
> x
[1] 5

> x <- 0
> while  (x < 5)
+ {
+   x <- x + 1;  if (x > 3) break
+ }
> x
[1] 4

> x <- 0
> repeat
+ {
+   x <- x + 1;  if (x >= 5) break
+ }
> x
[1] 5
```

More about for()

- `for (i in x) { }`:
  - If x has data file names, we can iteratively read the files corresponding to the names.
  - It is possible to loop over nonvector objects using `get()`.
  - `get()`: It takes as an argument the name of some object and returns the object of that name.

```
> x <- c('dat1.txt','dat2.txt'); z <- NULL
> for (i in x) {y = read.table(i);   z <- c(z,sum(y))}
> z
[1] 21 36

> a <- matrix(1,2,2); b <- array(2,dim=c(2,2,3)); z <- NULL
> for (i in c('a','b')) {y <- get(i);   z <- c(z,sum(y))}
> z
[1]  4 24
```

IF ELSE

- `if (`*condition*`) {`*statements1*`} else {`*statements2*`}`:
  - *condition* should return a single `T` or `F`.
  - If *condition* is true, *statements1* is performed. Otherwise, *statements2* is performed.
  - It can work as a function call (e.g., `y = if (`*condition*`)` *expr1* `else` *expr2*, where *expr1* & *expr2* could be function objects).

```
> r <- 3
> if (r==4)
+ {
+   x <- 1
+ } else {
+   x <- 3
+   y <- 5
+ }
> x; y
[1] 3
[1] 5
```

IF ELSE

```
> x <- c(5,7,2,9,10)
> y <- if (x[3] >= 3) x else x+2
> y
[1]  7  9  4 11 12

> z <- if (sum(x) > 30) mean(x) else sd(x)
> z
[1] 6.6
```

Boolean Operators for Scalar

- `x & y`: AND; `x | y`: OR; Logical vector.
- `x && y`: AND; `x || y`: OR; Logical scalar.
- If `x` and `y` are vectors, `&&` and `||` work for the first elements of `x` and `y`.

```
> x <- c(2,5,4); y <- c(1,7,3)
> x > 3 & y < 2
[1] FALSE FALSE FALSE
> x > 3 | y < 2
[1] TRUE TRUE TRUE

> x > 3 && y < 2
[1] FALSE
> x > 3 || y < 2
[1] TRUE

> x[1] > 3 && y[1] < 2
[1] FALSE
> x[1] > 3 || y[1] < 2
[1] TRUE
```

Logical Values (Review)

- Logical values `TRUE; T` & `FALSE; F` can be used in arithmetic expressions.
- In that case, `T` and `F` change to 1 and 0, respectively.

```
> x <- c(6,2,9,4,3)
> sum(x >= 3)
[1] 4
> (x[1] > 0) + (x[2] > 1) * (x[3] <10) * 2
[1] 3

> (1 < 2) == T
[1] TRUE
> (1 < 2) == 1
[1] TRUE
```

Default Values for Arguments in Function

- If you want to give default values for arguments in your function, specify default values when you define the function.

```
> avg <- function(x, na.rm = F)
+ {
+   if (na.rm == T) x <- x[!is.na(x)]
+   mean <- sum(x) / length(x)
+   return(mean)
+ }
>
> x <- c(1,3,7,5,NA,9)
> avg(x)
[1] NA
> avg(x, na.rm=T)
[1] 5
```

Return Values in Function

- `return()`:
  - The return values of a function can be any R objects.
  - Multiple results $\Rightarrow$ List object.
  - Even function objects can be return values.
  - You can avoid explicit calls to `return()` $\Rightarrow$ Not a good approach.

```
> avg <- function(x)
+ {
+   mean <- sum(x) / length(x)
+   mean
+ }
> x <- 1:20
> avg(x)
[1] 10.5
```

Return Values in Function

```
> summary.stat <- function(x)
+ {
+   result <- list(n=length(x), avg=mean(x),
+                  quartile=quantile(x,prob=c(0.25,0.5,0.75)))
+   return(result)
+ }
> z <- summary.stat(x); z
$n
[1] 20

$avg
[1] 10.5

$quartile
  25%   50%   75%
 5.75 10.50 15.25

> z$avg
[1] 10.5
```

Return Values in Function

```
> g <- function()
+ {
+   t <- function(x) return(x^2)
+   return(t)
+ }

> g()
function(x) return(x^2)
<environment: 0x118e9c48>
```

Functions are Objects

- Since functions are object with class `"function"`,
  - If you type the name of function, you can see the function body.
  - You can assign functions to other objects.
  - You can use functions as arguments of other functions.

```
> avg <- function(x)
+ {
+   mean <- sum(x) / length(x)
+   mean
+ }
> avg
function(x)
{
  mean <- sum(x) / length(x)
  mean
}
```

Functions are Objects

```
> f1 <- function(x,y) return(x*y)
> f2 <- function(x,y) return(x/y)
> f <- f1
> f(4,2)
[1] 8
> f <- f2
> f(4,2)
[1] 2

> g <- function(z,x,y) return(z(x,y))
> g(f1,4,2)
[1] 8
> g(f2,4,2)
[1] 2
```

Functions are Objects

```
> f1 <- function(x) return(sin(x))
> f2 <- function(x) return(sqrt(x^2+1))
> f3 <- function(x) return(x*x-1)
> plot(c(0,1),c(-1,1.5), type='n',xlab='X',ylab='Y')
> for(f in c(f1,f2,f3)) plot(f,0,1,add=T)
```