Note 10. Additional Techniques in Functions

Introduction to Statistical Programming

Chanmin Kim

Department of Statistics
Sungkyunkwan University

2022 Spring

Recursion

- Recursive function: A function that calls itself.
- To solve a problem by a recursive function `f()`:
  (1) Break the original problem into one or more smaller same problems.
  (2) Within `f()`, call `f()` on each of the smaller problem.
  (3) Within `f()`, piece together the results of (2) to solve the original problem.
- Potential drawbacks of recursion:
  - It is not easy to use recursion because it is very abstract (Inverse of proof by mathematical induction).
  - It requires large use of memory because it repeatedly defines functions.

Recursion

```
> # Factorial function
> rfact <- function(x)
+ {
+   if (x == 0) return (1)
+   else return (x * rfact(x-1))
+ }

> rfact(5)
[1] 120
> 1*2*3*4*5
[1] 120
```

Recursion

```
> # Sort function
> qsort <- function(x)
+ {
+   if (length(x) <= 1) return(x)  # Termination condition
+   element <- x[1]
+   partition <- x[-1]
+
+   x1 <- partition[partition < element]
+   x2 <- partition[partition >= element]
+
+   x1 <- qsort(x1)
+   x2 <- qsort(x2)
+   return(c(x1,element,x2))
+ }

> y <- c(4,7,2,9,7)
> qsort(y)
[1] 2 4 7 7 9
```

Replacement Functions

- Example:

```
> x <- 1:3
> names(x) <- c('a','b','c')
> x
a b c
1 2 3
```

- This example has no problem.
- However, it looks like it has something wrong because it assigns a value to the result of a function call.
- R actually executes the following code:

```
x = 'names<-'(x,value=c('a','b','c'))
```

Replacement Functions

- `'names<-'()` is a pre-defined function and this type of function is called 'replacement function'.

```
> getAnywhere('names<-')
A single object matching 'names<-' was found
It was found in the following places
  package:base
  namespace:base
with value

function (x, value)  .Primitive("names<-")
```

- When any statement in which the left side is not just a variable name is considered, a replacement function can be used. (i.e., situation like `f(x) <- y` $\equiv$ `'f<-'(x,value=y)`).

Replacement Functions

```
> # Indexing
> x <- c(5,7,1,4,5)
> x[3] <- 4; x
[1] 5 7 4 4 5

> x <- c(5,7,1,4,5)
> x <- '[<-'(x,3,value=4); x
[1] 5 7 4 4 5

> # E.g.,
> 'cutoff<-' = function(x, value)
+ {
+   x[x > value] <- Inf
+   x
+ }

> y <- c(88,45,200,30,150)
> cutoff(y) <- 100
> y
[1]  88  45 Inf  30 Inf
```

Your Own Binary Operators

- Your own binary operators: Just write a function whose name begins and ends with %.

```
> '%ab2%' = function(a,b) a*b+2
> 5 %ab2% 2
[1] 12

> '%dist%' = function(x,y) sqrt(sum(x^2 + y^2))
> x = c(2,3)
> y = c(5,2)
> x %dist% y
[1] 6.480741
```

Anonymous Functions

- Anonymous functions: Functions without the function name (typically functions with 1 line code).

```
> f <- function(x) x / sum(x)
> x <- matrix(1:12,4,3)
> apply(x,2,f)
     [,1]      [,2]      [,3]
[1,]  0.1 0.1923077 0.2142857
[2,]  0.2 0.2307692 0.2380952
[3,]  0.3 0.2692308 0.2619048
[4,]  0.4 0.3076923 0.2857143
>
> apply(x,2,function(x) x/sum(x))
     [,1]      [,2]      [,3]
[1,]  0.1 0.1923077 0.2142857
[2,]  0.2 0.2307692 0.2380952
[3,]  0.3 0.2692308 0.2619048
[4,]  0.4 0.3076923 0.2857143
```

... in Function Arguments

- ... (three-dots): A mechanism that allows variability in the arguments given to R functions (Technically it is ellipsis).

```
> f <- function(x)
+ {
+   mx <- mean(x)
+   sx <- sd(x)
+   return(list(mean=mx,sd=sx))
+ }

> x = c(3,6,NA,8,6)
> f(x)
$mean
[1] NA
$sd
[1] NA
```

... in Function Arguments

```
> f <- function(x,...)
+ {
+   mx <- mean(x,...)
+   sx <- sd(x,...)
+   return(list(mean=mx,sd=sx))
+ }

> f(x,na.rm=T)
$mean
[1] 5.75
$sd
[1] 2.061553
```