Introduction to Computer Programming with R (FOR 6934)

Qing Zhao (School of Forest Resources & Conservation, qing.zhao@ufl.edu)
Daijiang Li (Department of Wildlife Ecology & Conservation, dli1@ufl.edu)
Denis Valle (School of Forest Resources & Conservation, drvalle@ufl.edu)

Class thirteen

- Understand customized functions
- Make customized functions with loops and if else statement
- Save customized functions and reuse them

Functions can be used repeatedly

```
x <- 3
y <- 2 * x + 1
y
```

```
## [1] 7
```

```
x <- 12.5
y <- 2 * x + 1
y
```

```
## [1] 26
```

Define functions for convenience

```
x.to.y <- function(x) { # Determine the function's name and
                        # input variable(s) needed in the function
    y <- 2 * x + 1 # This is the same line I used before
    return(y) # Don't forget to return the result
}
x.to.y(x=3)
```

```
## [1] 7
```

```
x.to.y(x=12.5)
```

```
## [1] 26
```

General forms of customized functions

name.of.function <- function(input variable 1, . . . , input variable m) {
command 1
command 2
. . .
command k

return(output)
}

Sample code

Create a function for power function

```
power.function <- function(base, power) {
    y <- base ^ power
    return(y)
}
power.function(base=3, power=2)
```

```
## [1] 9
```

```
power.function(base=c(4,9), power=.5)
```

```
## [1] 2 3
```

Sample code

Have a look at your function

```
power.function
```

```
## function(base, power) {
##     y <- base ^ power
##     return(y)
## }
## <bytecode: 0x00000000144b6380>
```

Sample code

Create functions to calculate multiple things

```
math <- function(x1, x2) {
    y1 <- x1 * x2
    y2 <- x1^2 + x2^2
    return(list(y1=y1, y2=y2))
}

math(x1=3, x2=-1)
```

```
## $y1
## [1] -3
##
## $y2
## [1] 10
```

Sample code

Re-use the function

```
a1 <- 1:4
a2 <- c(1, -1, 1, -1)
b <- math(x1=a1, x2=a2)
b
```

```
## $y1
## [1]  1 -2  3 -4
##
## $y2
## [1]  2  5 10 17
```

Sample code

Create a funtion to return full names

```
full.name <- function(first.name, last.name) {
    z <- paste(first.name, last.name, sep=' ')
    return(z)
}

full.name(first.name='bruce', last.name='Wayne')
```

```
## [1] "bruce Wayne"
```

```
full.name(first.name='stephen', last.name='cuRRy')
```

```
## [1] "stephen cuRRy"
```

Sample code

Make the function smarter to correct things

```
full.name2 <- function(first.name, last.name) {
    a1 <- substr(first.name, 1, 1)
    a2 <- substr(first.name, 2, 1000)
    first.new <- paste(toupper(a1), tolower(a2), sep='')
    b1 <- substr(last.name, 1, 1)
    b2 <- substr(last.name, 2, 1000)
    last.new <- paste(toupper(b1), tolower(b2), sep='')
    z <- paste(first.new, last.new, sep=' ')
    return(z)
}
```

Sample code

And use it

```
full.name2(first.name='bruce', last.name='Wayne')
```

```
## [1] "Bruce Wayne"
```

```
full.name2(first.name='stephen', last.name='cuRRy')
```

```
## [1] "Stephen Curry"
```

This concludes Class 13, Section 1

Please continue on to the next video

Make functions with loops and if else statement

- Loops and ifelse statement are useful, but time-consuming to write
- Make functions to re-use them easily

Sample code

Generate some random points with latitude and longitude values

```
lat <- rnorm(4, mean=0, sd=1)
lon <- rnorm(length(lat), mean=0, sd=1)
cbind(lat, lon)
```

```
##              lat        lon
## [1,] -0.04015754  0.1141017
## [2,] -1.10159784  0.9490788
## [3,]  0.51507828  0.3556001
## [4,]  2.34940653 -1.0055387
```

Sample code

Use loops to calculate distance

```
dist <- matrix(, nrow=length(lat), ncol=length(lon))
for (i in 1:length(lat)) {
    for (j in 1:length(lon)) {
        dist[i,j] <- sqrt((lat[i]-lat[j])^2 + (lon[i]-lon[j])^2)
    }
}
dist
```

```
##           [,1]     [,2]      [,3]     [,4]
## [1,] 0.0000000 1.350497 0.6054819 2.638866
## [2,] 1.3504970 0.000000 1.7221668 3.966101
## [3,] 0.6054819 1.722167 0.0000000 2.284176
## [4,] 2.6388655 3.966101 2.2841758 0.000000
```

Sample code

Copy-paste the same code when changing latitude and longitude

```
lat <- rnorm(3, mean=5, sd=2)
lon <- rnorm(length(lat), mean=-2, sd=2)
# I copy-paste the rest part
dist <- matrix(, nrow=length(lat), ncol=length(lon))
for (i in 1:length(lat)) {
    for (j in 1:length(lon)) {
        dist[i,j] <- sqrt((lat[i]-lat[j])^2 + (lon[i]-lon[j])^2)
    }
```

```
}
dist
```

```
##          [,1]    [,2]     [,3]
## [1,] 0.000000 2.99918 2.513425
## [2,] 2.999180 0.00000 4.313520
## [3,] 2.513425 4.31352 0.000000
```

Sample code

Make the code a function so that we can easily re-use it

```
distance <- function(lat, lon) { # define the name of the function and the variables needed
    # Here I simply copy-paste the above code
    dist <- matrix(, nrow=length(lat), ncol=length(lon))
    for (i in 1:length(lat)) {
        for (j in 1:length(lon)) {
            dist[i,j] <- sqrt((lat[i]-lat[j])^2 + (lon[i]-lon[j])^2)
        }
    }
    return(dist) # Don't forget to return the result
}
```

Sample code

Now I can easily re-use it

```
distance(lat=rnorm(6, 1, .5), lon=rnorm(6, -2, 1))
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.0000000 1.8627746 0.9277088 0.7522218 0.7309213 0.5471728
## [2,] 1.8627746 0.0000000 0.9672987 1.2729720 1.3238560 1.6175541
## [3,] 0.9277088 0.9672987 0.0000000 0.6145534 0.6606648 0.6626179
## [4,] 0.7522218 1.2729720 0.6145534 0.0000000 0.0529863 0.8990614
## [5,] 0.7309213 1.3238560 0.6606648 0.0529863 0.0000000 0.9113909
## [6,] 0.5471728 1.6175541 0.6626179 0.8990614 0.9113909 0.0000000
```

Sample code

Use loops and if else statement to calculate correlation matrix

```
cov2cor <- function(cov) {
    if (isSymmetric(cov)) {
        cor <- matrix(, nrow=dim(cov)[1], ncol=dim(cov)[2])
        for (i in 1:dim(cov)[1]) {
            for (j in 1:dim(cov)[2]) {
                cor[i,j] <- cov[i,j] / sqrt(cov[i,i]) / sqrt(cov[j,j])
            }
        }
        note <- 'A covariance matrix is converted to a correlation matrix'
    } else {
        cor <- NA
        note <- 'The input matrix is not a covariance matrix'
```

```
    }
    return(list(cor=cor, note=note))
}
```

Use the function

```
cov1 <- matrix(c(2,1.5,1.5,3), nrow=2, ncol=2)
cov2cor(cov1)
```

```
## $cor
##           [,1]      [,2]
## [1,] 1.0000000 0.6123724
## [2,] 0.6123724 1.0000000
##
## $note
## [1] "A covariance matrix is converted to a correlation matrix"
```

Re-use it again

```
cov2 <- matrix(c(2,.6,1.6,3), nrow=2, ncol=2)
cov2cor(cov2)
```

```
## $cor
## [1] NA
##
## $note
## [1] "The input matrix is not a covariance matrix"
```

Comments

- When you have some code that you want to re-use, you can make them functions
- Define a function name and the variables that are needed in the function
- Copy the code you want to re-use in the function (sometimes you need to make changes)
- Remember to return the results

This concludes Class 13, Section 2

Please continue on to the next video

Use customized functions within other customized functions

- Create a function for spatial smoothing

Sample code

A function to calculate distance was created

```
distance
```

```
## function(lat, lon) { # define the name of the function and the variables needed
##     # Here I simply copy-paste the above code
##     dist <- matrix(, nrow=length(lat), ncol=length(lon))
##     for (i in 1:length(lat)) {
##         for (j in 1:length(lon)) {
##             dist[i,j] <- sqrt((lat[i]-lat[j])^2 + (lon[i]-lon[j])^2)
##         }
##     }
##     return(dist) # Don't forget to return the result
## }
## <bytecode: 0x000000001939db50>
```

Sample code

Use the distance function in a spatial smoothing function

```
smooth <- function(lat, lon, temp) {
    dist <- distance(lat, lon)
    temp_new <- numeric(length(temp))
    for (i in 1:length(temp)) {
        weight <- 1 / (dist[,i] + 1)
        temp_new[i] <- sum(temp * weight) / sum(weight)
    }
    return(temp_new)
}
```

Sample code

Use the code to conduct spatial smoothing

```
n <- 10
lat <- rep(1:n, each=n)
lon <- rep(1:n, times=n)
temp <- rnorm(n^2, 0, 5)
temp_new <- smooth(lat, lon, temp)
temp_old_new <- cbind(temp, temp_new)
apply(temp_old_new, 2, mean)
```

```
##       temp   temp_new
## -0.9330835 -1.0430082
```

```
apply(temp_old_new, 2, sd)
```

```
##     temp temp_new
## 4.995949 0.353359
```

Sample code

Even more, simulate data and do spatial smoothing

```r
sim.and.smooth <- function(n) {
    lat <- rep(1:n, each=n)
    lon <- rep(1:n, times=n)
    temp <- rnorm(n^2, 0, 5)
    temp_new <- smooth(lat, lon, temp)
    temp_old_new <- cbind(temp, temp_new)
    temp_mean <- apply(temp_old_new, 2, mean)
    temp_sd <- apply(temp_old_new, 2, sd)
    out <- data.frame(Mean=temp_mean, SD=temp_sd)
    return(out = out)
}
```

Sample code

Let's try it

```r
sim.and.smooth(20)
```

```
##                Mean        SD
## temp     0.04119570 4.9930483
## temp_new 0.03088137 0.1660674
```

Save and re-used customized functions

- Save R script as a .R file
- Use source() function to load the saved functions

Save R script as a .R file

- Create a R script and put your functions in it
  - "File" -> "New script" in RGui
  - "File" -> "New file" -> "R script" in RStudio
- Save your functions
  - "File" -> "Save" in RGui
  - "File" -> "Save" in RStudio

Sample code

Load your functions

```r
source(file='c:/data/spatial.R')
```

Sample code

Use the loaded code

```r
sim.and.smooth(20)
```

```
##                Mean        SD
## temp     0.4733164 5.0636203
## temp_new 0.4553918 0.2075167
```

Summary

- Learn to create customized functions
- Create customized functions to re-use long code with loops and if else statement
- Use customized functions in customized functions
- Save customized functions for later use

Thank you and see you next class