R computing for Business Data Analytics

Instructor: Dr. Howard Hao-Chun Chuang
Assistant Professor, Department of Management Information Systems
National Chengchi University, Taipei, Taiwan 116
chuang@nccu.edu.tw

Last Revised: August 2014

2.1 Writing *R* functions

- What are functions?

If you find yourself running the same code repeatedly, you should consider turning the code into a function for the sake of efficiency and maintainability. Generally, functions take inputs, process the inputs (e.g., calculation, graph drawing), and produce outputs.

Let's start with a simple function that takes merely one input.

```
> double.num = function(x){  #x is the argument of this function
> x*2}
> double.num(5)
```

Tweak the function a bit with an explicit *return*.

```
> double.num = function(x){
> return(x*2)
> print("Hello")
> return(17)}
> double.num(5)
```

What happens now? Do we get to see all outputs?

```
> oddcount = function(x){
> k=0
>     for(n in 1:length(x)){
>         if(x[n]%%2==1) k=k+1
>     }
> return(k)
> }
> oddcount(0:10)
```

- In-class exercise

Suppose payments of R dollars are deposited annually into a bank account that earns constant interest i per year (assuming deposits are made at the end of each year). The total amount at the end of n years is

$$R(1+i)^{n-1}+...+R(1+i)+R=R\frac{(1+i)^n-1}{i}$$

Please write a function (named as "annuityAmt") to calculate the amount of an annuity. The function is supposed to have three arguments – R, i, and n. If $400 is deposited annually for 10 years into an account bearing 5% annual interest, what is the accumulated amount? (The function should return 5031.157)

2.2 Control statements

- *If* and *else*

The syntax for *if* is

if (condition) {commands when TRUE}

```
> toCheck=1
> if (toCheck ==1){
> print("hello")
> }

> if (toCheck==0){
> print("hello")
> }
```

The syntax for *if…else* is

```
if (condition) {
   commands when TRUE
} else {
   commands when FALSE
}
```

Let's write a function.

```
> check.bool=function(x){
> if (x ==1){
>     print("hello")
> } else {
>     print("goodbye") }
> }
> check.bool(1)
> check.bool(0)
```

- *Ifelse*

While *if* is more like the *if* statement in traditional programming languages, *ifelse* is more like the *if* function in Excel.

```
> ifelse(1==1, "Yes", "No")

> toTest=c(1, 1, 0, 1, 0, 1)
> ifelse(toTest==1, toTest*3, "Zero")
```

We can try a compound test.

```
> a=c(1, 1, 0, 1)
> b=c(2, 1, 0, 1)
> ifelse(a==1 & b==1, "Yes", "No")
```

2.3 Loops

- *For* loops

The syntax for *for* is

for (name in vector) {commands}

```
> for (i in 1:10){
>      print(i)
> }
```

or

```
> print(1:10)
```

```
> fruit = c("apple", "orange", "watermelon")
> fruitL=rep(NA, length(fruit))
> names(fruitL)=fruit
> for (i in 1:length(fruit)){
>       fruitL[i]=nchar(fruit[i])
> }
> fruitL
```

- *While* loops

The syntax for *while* is

while (condition) {statements}

```
> x=1
> while (x<=5){
>          print(x)
>          x=x+1
> }
```

We can always rewrite a *for* loop as a *while* loop.

- Controlling loops

Sometimes we put *next* and/or *break* into for( ) and while( ) loops.

next

```
> for (i in 1:10){
>       if(i==3){next}
>       print(i)
> }
```

break

```
> for (i in 1:10){
>       if(i==4){break}
>       print(i)
> }
```

2.4 Recursion

- Newton's method for root finding

This is a popular numerical method to find a root of an algebraic equation: $f(x)=0$. If $f(x)$ has derivative $f'(x)$, the following iteration will converge to a root of the above equation.

$$x_0 = \text{initial guess}$$
$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Suppose $f(x)=x^3+2x^2-7$, we can the root for $f(x)=0$ using the following recursion.

$$x_n = x_{n-1} - \frac{x_{n-1}^3 + 2x_{n-1}^2 - 7}{3x_{n-1}^2 + 4x_{n-1}}$$

How do we implement this recursive algorithm in *R*?

```
> x = x0 #x0 is an arbitrary guess
> f=x^3+2*x^2-7
> tolerance=0.000001
> while(abs(f) > tolerance){
>       f.prime=3*x^2+4*x
>       x=x-f/f.prime
>       f=x^3+2*x^2-7
> }
> x
```

Recursion is a powerful trick that makes our life much easier. We can re-write the example above using a *repeat* loop + a *break* statement.

```
> x = x0 #x0 is an arbitrary guess
> tolerance=0.000001
> repeat{
>       f=x^3+2*x^2-7
>       if(abs(f) < tolerance) break
>       f.prime=3*x^2+4*x
>       x=x-f/f.prime
> }
> x
```