R – Logics and Automation

Lecture 6

- Controlling the logic flow
- Iteration loop
- “FOR” loop
- If loop
- Nested looping

Looping

Looping allows to run the command many times. For example, if one needs to print a sentence, saying “I like Prince George”, it can be simply done by the command

```
print(“I like Prince George”)
```

Looping

Now, if the sentence is required to repeat 10 times, how to do that?

One simple but awkward way is to repeat the above command 10 times.

Is there any “smarter” way? Yes, there is a specific loop called “for” loop.

“for” Loop in R

Conceptually, a loop is a way to repeat a sequence of instructions under certain conditions. They allow you to automate parts of your code that are in need of repetition.

It will become more clear to you once you start working with some examples below.

```
for (index in 1:10){
print(“I love Prince George”)}
```

“for” Loop in R

To understand this code, we need to learn a basic grammar of “for” loop.

- The key word “for”, followed by parentheses;
- An identifier (called index) between the parentheses.
- The key word “in”, which follows the identifier;
- A vector with values to loop over;
- A code block between brace that has to be carried out for every value in the vector;

“for” Loop in R

The code block itself is not controlled by the value of identifier (index). Often, the code itself is controlled by the identifier. For example, we want to print calendar year from 2010 to 2015 respectively, after the sentence “This year is”. Still you can use the awkward way like

```
print(paste("The year is", 2010))
print(paste("The year is", 2011))
print(paste("The year is", 2012))
print(paste("The year is", 2013))
print(paste("The year is", 2014))
print(paste("The year is", 2015))
```

“for” Loop in R

Or you can use “for’ loop for a smart way

```
for (i in 2010:2015){
print(paste(“This year is”, i))
}
```

One also can use other vector expression for this purpose, i..e,

```
x <- c(2010,2011,2012,2013,2014,2015)
for(i in x){print(paste(“this year is”, x))}
```

“for” Loop in R

You will have the output like

```
[1] "this year is 2010" "this year is 2011" "this year is 2012"
[4] "this year is 2013" "this year is 2014" "this year is 2015"
[1] "this year is 2010" "this year is 2011" "this year is 2012"
[4] "this year is 2013" "this year is 2014" "this year is 2015"
[1] "this year is 2010" "this year is 2011" "this year is 2012"
[4] "this year is 2013" "this year is 2014" "this year is 2015"
```

“for” Loop in R

This is not what we expect, what is wrong here? This is because you print all vector x at each iteration of the loop. The correct way should be:

```
x<-c(2010,2011,2012,2013,2014,2015)
for(i in x){print(paste("this year is", i))}
```

Or

```
for(i in seq(2010,2015,by=1)){print(paste("this year is",i))}
```

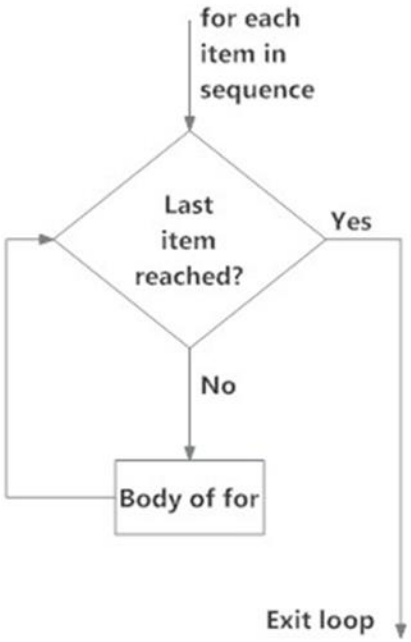
“for” Loop in R

Or

```
for(i in 2010:2015){print(paste("this year is", i))}
```

Output is

```
[1] "this year is 2010"
[1] "this year is 2011"
[1] "this year is 2012"
[1] "this year is 2013"
[1] "this year is 2014"
[1] "this year is 2015"
```



“for” Loop in R

For another example, we want to print 1, 4, 9, 16, 25. It can be coded as below

```
for(i in 1:5){print(i^2)}
```

If we have a vector (-3, 6,2,5,9), and wish to print the square of its each element, we can write

```
for(i in c(-3, 6, 2, 5, 9)){print(i^2)}
```

Or

```
x<-c(-3, 6, 2, 5, 9)
for(i in x){print(i^2)}
```

“for” Loop in R

If we want to store the above solution in a vector for future use, we can do as below

```
yy<-numeric(5)
x<-c(-3, 6, 2, 5, 9)
m<-0
for(i in x){
m<- m+1
yy[m]<-i^2
}
```

Or

```
yy<-numeric(5);
for(i in 1:5){yyy[i]<-x[i]^2};
```

Here we use array (vector) to manipulate the process, i.e., in “for loop”, the identifier is the “room number”, not “room” itself.

Example

We have the temperature record in degree C of (-3, 6, 2, 5, 9), lets transfer them in degree F.

We know the transfer formula by

Degree F = Degree C * 9/5 + 32

We can code as below

```
x<- c(-3,6,2,5,9)
y<- numeric(5)
for(i in 1:5){
y[i]<- x[i]*9/5+32
}
```

Example 4.1 (text book)

To generate a Fibonacci sequence that has the pattern [1, 1, 2, 3, 5, 8, …..], i.e., subsequent elements are defined as the sum of the preceding two elements.

```
x<-numeric(12)
x[1] <- 1
x[2] <- 1
for(i in 3:12){
x[i]<- x[i-1] + x[i-2]
}
x
```

Output: 1 1 2 3 5 8 13 21 34 55 89 144

Example

```
u1 <- rnorm(30)
print("This loop calculates the square of the first 10
elements of vector u1")

# Initialize `usq`
usq <- 0

for(i in 1:10) {
  # i-th element of `u1` squared into `i`-th position of
`usq`
  usq[i] <- u1[i]*u1[i]
  print(usq[i])
}

print(i)
```


Source: www.datacamp.com

Nested “For loop”

We have discussed the basic structure and control flow of the “for loop” and demonstrated some examples. In some cases, there are multiple variables to control the flow, in which we may need the multiple loops, or, nested “for” loop. The grammar is as below

```
for(index1 in vector1){
for(index2 in vector2){
  code1}
{code2}
}
```

Example

For example, one vector is x=[2, 3,-2,4,5], the other vector is y=[-1, 2, 3,5,6]; we calculate their cross-product, which should be a matrix;

```
z = matrix(nrow=5,ncol=5)
x<- c(2,3,-2,4,5);
y<- c(-1,2,3,5,6)
 for (i in 1:5) {
    for(j in 1:5){
    z[i,j]=x[i]*y[j];
    }
 }
```

Example

Generate the 5x 5 matrix A, whose (i; j) entry is i+j

```
z<- matrix(nrow=5,ncol=5);
for(i in 1:5){
  for(j in 1:5){
  z[i,j]<- i+j
  }
}
```

Example

```
# Create a 30 x 30 matrix (of 30 rows and 30 columns)
mymat <- matrix(nrow=30, ncol=30)

# For each row and for each column, assign values based
on position: product of two indexes
for(i in 1:dim(mymat)[1]) {
  for(j in 1:dim(mymat)[2]) {
    mymat[i,j] = i*j
  }
}

# Just show the upper left 10x10 chunk
mymat[1:10, 1:10]
```


Source: www.datacamp.com

“if” loop

Decision making is an important part of programming. For example, one plans to fish if it is sunny. Intuitionally we have the below command

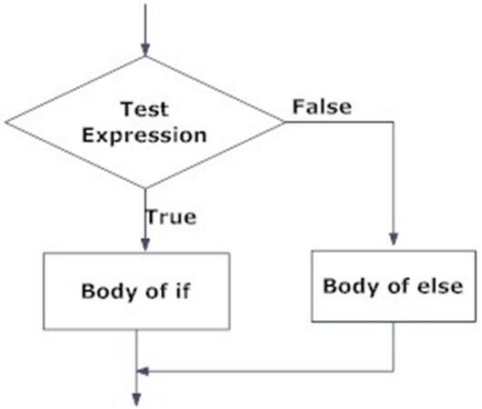
If (weather = sunny) {fishing}

Generally, such kinds of tasks can be achieved in R programming using the conditional “if...statement”

“if” loop

The syntax of if statement is:

```
if (condition) {
   statement
}
```



If the condition is TRUE, the statement gets executed. But if it's FALSE, nothing happens. Here, condition can be a logical or numeric vector, but only the first element is taken into consideration.

In the case of numeric vector, zero is taken as FALSE, rest as TRUE.

“if” loop Example

```
x <- 5
if(x > 0){
   print("Positive number")
}
```

Output

"Positive number"

“if … else” statement

“If there are two conditions to be applied, we will use “if...else statement”,

The syntax of if...else statement is:

```
if (condition) {
   statement1
} else {
   statement2
}
```

“if … else” statement

Example:

```
x <- -5
if(x > 0){
   print("Non-negative number")
} else {
   print("Negative number")
}
```

Output: "Negative number"

Nested “if - else” statement

If there are more conditions to be applied, one can use “Nested if...else statement”. In fact, we can nest as many “if...else statement” as we want as follows.

The syntax of nested if...else statement is:

```
if ( condition1) {
   statement1
} else if ( condition2) {
   statement2
} else if ( condition3) {
   statement3
} else {
   statement4}
```

Nested “if - else” statement

```
x <- 0
if (x < 0) {
   print("Negative number")
} else if (x > 0) {
   print("Positive number")
} else
   print("Zero")
```

Output "Zero"

In “if”, “if…else”, or “nested if”, only one statement will get executed depending upon the condition. After the statement is executed, the “if…” loop terminates, and the programming moves to the first statement just after the “if …” loop.

Basic Logical Operations

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
= =	Equal to exactly
! =	Not equal to
|	Or (vectorized）
||	Or (first element)
&	And (vectorized)
&&	And(first element)

Examples

1<2 -- true
5 = =10 --- false
5 !=10 ---true

x<-seq(2, 20,2)
x==2 | x>=8
Output: TRUE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

x==2 || x>=8
Output: True

Examples

```
x==2 & x>=8
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
x==2 && x>=8
[1] FALSE
n1 <-0
n2 <-0
if(n1>0 & n2>0){print("positive product")}else
  if(n1<0 & n2<0){print(“negative product")} else
    if(n1==0 | n2 ==0){print("zero produce")}else
{print("negative product")}
```

Example 4.5 (text book)

A prime number is a number greater than 1 that cannot be formed by multiplying two smaller numbers. To list prime number up to a give value n. For example, if n =10, the prime number is 2,3,5,7. We use “if statement” for the programming.

```
n<- 100
x<- seq(2,n)
primes<- c()
for(i in 2:n){
  if (any(x==i)){
  primes<- c(primes, i)
  x<- c(x[(x%%i)!=0],i)}
}
```

Example 4.5 (text book)

Coding interpretation

(x%%i)!=0 --- true if the remainder of x divided by i is not 0, otherwise false.

x[(x%%i)!=0] --- removing all numbers that can be divided by i from x;

Example 4.5 (text book)

For example

originally, x=[2,3,4,5,6,7,8,9,10] lets say i = 2

x[(x%%2)! = 0] -> x = [3,5,7,9]

x<-c(x[(x%%2)!=0],2) -> producing a vector x= [3,5,7,9,2];

primes<-c(primes,2) -> put 2 in the vector of primes

if (any(x==i))

this is an important condition that ignores all number of multiples of i in reconstructing x and listing prime. For example, if i=2, this condition will not consider 4,6,8,…, which is correct. If this condition is removed, we will reconstruct x that is the same as original one, and list all numbers of x in the vector of prime, which is not correct.

Example wordcount.R

```
words <- c("it","was","the","dirty","end","of","winter")

#loop over the words
for ( w in words ) {

  w.length <- nchar( w )     # calculate the number of letters
  W <- toupper( w )          # convert the word to upper case letters
  msg <- paste( W, "has", w.length, "letters" )   # a message to print
  print( msg )               # print it

}
```

Source: learningstatisticswithr.com

Exercise

- Download the enso_index.csv file from course website i.e. SST data for ENSO
  - http://web.unbc.ca/~islam/ENSC250/data.html
- Load the file in R notebook and select YEAR and NINO3.4 columns for analysis
- Variable NINO3.4 is monthly time series for 24 years (1990-2013). So the total values are 288.
- Redesign variable NINO3.4 into 12 by 24 matrix representing 12 months in a year and 24 years in total.
  - Hint : nino_mat <- matrix(NINO3.4, nrow=12, ncol=24)
- Then use "for"" loops to subtract mean of each month from corresponding monthly value.
  - Hint : Define an empty matrix first i.e. "anom" having same dimension as "nino_mat" matrix but with all values zeros. During looping, update each index of the matrix.
- Plot the monthly mean data as shown in next slide.

Exercise: Output