TMC 105
Statistical Data Analysis with R
Unit 3
R Flow Control

Presented By : Aditya Joshi

Asst. Professor

Department of Computer Application

Graphic Era Deemed to be University

R if…else Statement

Decision making is an important part of programming. This can be achieved in R programming using the conditional if...else statement.

R if statement

The syntax of if statement is:

```
if (test_expression) {
statement
}
```

if the test_expression is TRUE, the statement gets executed. But if it’s FALSE, nothing happens.

Here, test_expression 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.

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

if…else statement

The syntax of if…else statement is:

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

The else part is optional and is only evaluated if test_expression is FALSE.

It is important to note that else must be in the same line as the closing braces of the if statement.

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

The above conditional can also be written in a single line as follows.

```r
if(x > 0) print("Non-negative number") else print("Negative number")
```

This feature of R allows us to write construct as shown below.

```
> x <- -5
> y <- if(x > 0) 5 else 6
> y
[1] 6
```

if…else Ladder

The if…else ladder (if…else…if) statement allows you execute a block of code among more than 2 alternatives

The syntax of if…else statement is:

```
if ( test_expression1) {
statement1
} else if ( test_expression2) {
statement2
} else if ( test_expression3) {
statement3
} else {
statement4
}
```

example

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

R ifelse() Function

- **This is a shorthand function to the traditional if…else statement.**

Vectors form the basic building block of R programming.

Most of the functions in R take vector as input and output a resultant vector.

This vectorization of code, will be much faster than applying the same function to each element of the vector individually.

Similar to this concept, there is a vector equivalent form of the if…else statement in R, the ifelse() function.

Syntax of ifelse() function

ifelse(test_expression, x, y)

Here, test_expression must be a logical vector (or an object that can be coerced to logical). The return value is a vector with the same length as test_expression.

This returned vector has element from x if the corresponding value of test_expression is TRUE or from y if the corresponding value of test_expression is FALSE.

Example: ifelse() function

```
> a = c(5,7,2,9)
> ifelse(a %% 2 == 0,"even","odd")
[1] "odd"  "odd"  "even" "odd"
```

In the above example, the test_expression is a %% 2 == 0 which will result into the vector (FALSE,FALSE,TRUE ,FALSE).

R for Loop

Loops are used in programming to repeat a specific block of code.

A for loop is used to iterate over a vector in R programming.

Syntax of for loop

```
for (val in sequence)
{
statement
}
```

Here, sequence is a vector and val takes on each of its value during the loop. In each iteration, statement is evaluated.

Example: for loop

Below is an example to count the number of even numbers in a vector.

```
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x) {
if(val %% 2 == 0)  count = count+1
}
print(count)
```

In the above example, the loop iterates 7 times as the vector x has 7 elements.

In each iteration, val takes on the value of corresponding element of x.

We have used a counter to count the number of even numbers in x. We can see that x contains 3 even numbers.

Check Positive, Negative or Zero number

```
# In this program, we input a number check if the number is positive or
negative or zero
num = as.double(readline(prompt="Enter a number: "))
if(num > 0) {
print("Positive number")
} else {
if(num == 0) {
print("Zero")
} else {
print("Negative number")
}
}
```

Check Odd and Even Number

```
# Program to check if the input number is odd or even.
# A number is even if division by 2 give a remainder of 0.
# If remainder is 1, it is odd.
num = as.integer(readline(prompt="Enter a number: "))
if((num %% 2) == 0) {
print(paste(num,"is Even"))
} else {
print(paste(num,"is Odd"))
}
```

Multiplication Table

```
# R Program to find the multiplicationtable (from 1 to 10)
# take input from the user
num = as.integer(readline(prompt = "Enter a number: "))
# use for loop to iterate 10 times
for(i in 1:10) {
print(paste(num,'x', i, '=', num*i))
}
```

R Program to Find the Factorial of a Number

We can also use the built-in function factorial() for this.

```
# take input from the user
num = as.integer(readline(prompt="Enter a number: "))
factorial = 1
# check is the number is negative, positive or zero
if(num < 0) {
print("Sorry, factorial does not exist for negative numbers")
} else if(num == 0) {
print("The factorial of 0 is 1")
} else {
for(i in 1:num) {
factorial = factorial * i
}
print(paste("The factorial of", num ,"is",factorial))
}
```

Check Prime Number

A positive integer greater than 1 which has no other factors except 1 and the number itself is called a prime number.

Numbers 2, 3, 5, 7, 11, 13 etc. are prime numbers as they do not have any other factors.

But, 6 is not prime (it is composite) since, 2 x 3 = 6.

```
# Program to check if the input number is prime or not
# take input from the user
num = as.integer(readline(prompt="Enter a number: "))
flag = 0
# prime numbers are greater than 1
if(num > 1) {
# check for factors
flag = 1
for(i in 2:(num-1)) {
if ((num %% i) == 0) {
flag = 0
break
}
}
}
if(num == 2)    flag = 1
if(flag == 1) {
print(paste(num,"is a prime number"))
} else {
print(paste(num,"is not a prime number"))
}
```

R while Loop

Loops are used in programming to repeat a specific block of code.

In R programming, while loops are used to loop until a specific condition is met.

Syntax of while loop

```
while (test_expression)
{
statement
}
```

Here, test_expression is evaluated and the body of the loop is entered if the result is TRUE.

The statements inside the loop are executed and the flow returns to evaluate the test_expression again.

This is repeated each time until test_expression evaluates to FALSE, in which case, the loop exits.

```
i <- 1
while (i < 6) {
print(i)
i = i+1
}
```