manipalglobal
Academy of Data Science

Control Structures

Table of Content

Introduction 3
1. Control Structures 4
1.1 If-else 4
1.2 Loops (using “for” loops) 5
1.3 While Loops 6
Application of Loops on Data frames 9
Nested While Loop 11
1.4 Repeat 12
1.5 Next 13
1.6 Return 14
Summary 15

Introduction

This topic deals with control structures and conditional statements in R programming, which allows the programmer to take control of the flow of execution of the program using loops and if-else blocks. For, while, and repeat loops are discussed in brief. The need for if-else blocks and nested if-else blocks are demonstrated through programming.

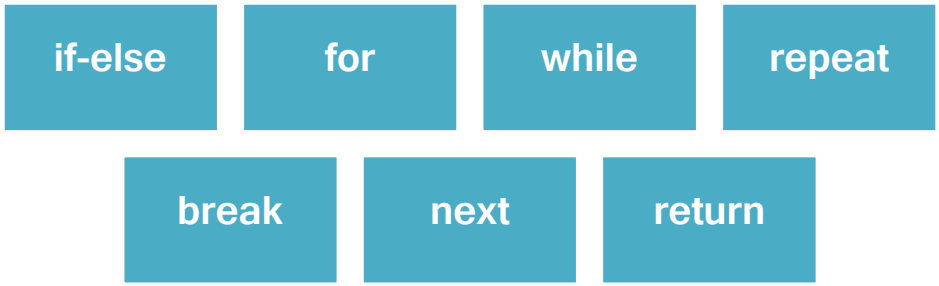
Learning Objectives

Upon completion of this topic, you will be able to:

- Understand the use of loops
- Demonstrate the ability to program using loops
- Identify the scenario to use For loops or While loops
- Using the if-else blocks in the R program
- Understand the purpose of using nested if-else blocks

1. Control Structures

Control structures allow you to control the flow of execution of a program. A few common control structures are:



1.1 If-else

In the if-else block, if the condition is TRUE, the code block following the “if” condition gets executed. In case of failure of the condition, the code following the “else” block gets executed.

Pseudocode:

```
if (condition)
{ # do something
}
else
{ # do something else
}
```

Example:

```
x =5
if(x > 0)
{   print("Non-negative number")
} else #Comment, the else statement must begin immediately after the }
{   print("Negative number")
}
```

Output:

```
[1] "Non negative number"
```

1.2 Loops (using "for" loops)

A "for" loop works on an iterable variable and assigns successive values till the end of a sequence. It repeats the same task for changing values of a given variable.

Pseudocode:

```
for (var in vectorValues)
{   #execute block of code
}
```

Example: Demonstrate the use of "for-loop" which prints a set of even numbers between 1 to 10.

```
for(i in c(1:10)) {
  if(i%%2==0){  #The double %% symbol is the modulo operator
 #i %% 2 == 0 means, if the remainder of a number divided by 2 is zero
     print(i)
 } #end of if block
} #end of for block
```

Output:

```
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

1.3 While Loops

While loops repeat the same operation until a test condition is met.

Pseudocode:

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

In the above pseudocode, 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 re-evaluate the test_expression. This is repeated each time until test_expression evaluates to FALSE, and the loop exits.

Example 1: Print a list of numbers starting from 0 to 10. If the counter variable exceeds the number 10, the control would exit the loop.

```
i=0
while(i<=10)
 {
print(i)
  i=i+1
}
```

Output:

```
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Example 2: The output of example 1 can be printed in the same line using the cat() function.

```
i=0
while(i<10) {
cat(i, " ") # the quotes are included in the command to print the output
#with white spaces
  i=i+1
}
```

Output:

```
0  1  2  3  4  5  6  7  8  9  10
```

Example 3: The program demonstrates how to print fibonacci numbers using while loops.

```
i=0
fib1=-1
fib2=1
while(i<10){
  fibseries=fib1+fib2
  cat(fibseries," ")
   i=i+1
   fib1=fib2
   fib2=fibseries
}
```

Output:

```
0 1 1 2 3 5 8 13 21 34
```

Example 4: The program demonstrates how to print the sum of squares of numbers in a vector.

```
i=1
num=0
a=c(1,2,3,4,5,6,7,8,9,10)
len=length(a)
while(i<=len){
  num=num + (a[i]*a[i])
  i=i+1
}
print(num)
```

Output:

```
[1] 385
```

Application of Loops on Data frames

Example: The program demonstrates Loops through the built-in iris dataset which creates a new variable Speciesstatus and assigns a numeric value based on the different status of the species in the dataset.

```
unique(iris$Species)

for(i in 1:nrow(iris)){
  if(iris$Species[i]=="setosa"){
  iris$Speciesstatus[i] <- 0
}else if(iris$Species[i]=="versicolor")
```

```
{
  iris$Speciesstatus[i] <- 1
}else iris$Speciesstatus[i] <-2
}
Head(iris) # Head() function would print the first 6 rows from a data
frame
```

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Speciesstatus
1	5.1	3.5	1.4	0.2	setosa	0
2	4.9	3.0	1.4	0.2	setosa	0
3	4.7	3.2	1.3	0.2	setosa	0
4	4.6	3.1	1.5	0.2	setosa	0
5	5.0	3.6	1.4	0.2	setosa	0
6	5.4	3.9	1.7	0.4	setosa	0

A new column speciesstatus has been added to the data frame iris.

Nested While Loop

A nested loop is a loop within a loop. The innermost loop is called an inner loop within the context of an outer loop.

Example 1: The program demonstrates how to print the numeric pattern.

1 1 1 1

1 1 1 1

1 1 1 1

Count the number of rows - its 3.

Count the number of elements in the each row - its 4.

The first element to be printed is 1.

Initialise, *i=1*

```
rows = 3
max = 4
ol = 1  # This is an initialiser to the outer loop
while(ol<=rows) # This loop will execute 3 times (the outer loop)
{       while(i <= max )  { # This loop will print 1 1 1 1 (the inner
loop)
    cat(1," ")
    i=i+1  }
  cat("\n") # This will print a new line each time 1 1 1 1 is printed
  i=1 # re-initialise i=1 so that the next line is 1 1 1 1
 ol=ol+1
}
```

Example 2: A simple loop to print the following pattern:

1 2 3 4 5

6 7 8 9 10

11 12 13 14 15

```
v=1:15
count=0
for(i in v){
  cat(i,"")
  count=count+1
  if(count%%5==0){cat("\n")}
}
```

1.4 Repeat

Repeat statement iterates a block of code multiple times until a break() statement is encountered, else the loop continues to execute infinitely eventually leading a program to crash.

Pseudocode:

```
repeat {
   commands …
   if(condition) {
      break
   }
}
```

Example:

```
i=0
repeat { # Start of the repeat block
  print(i)
  if(i==10) {break}  # Break statement is used to exit the loop
  i=i+1
} # This is the end of the repeat block
```

Output:

```
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

In this example, once "i" takes the value of 10, the control exists the loop.

1.5 Next

A Next statement is useful when the current iteration of a loop has to be skipped without terminating it. On encountering next(), the R parser skips further evaluation and starts next iteration of the loop.

When the variable “val” takes the value 3, the loop skips the current evaluation of statements and starts the next iteration of the loop.

Example:

```
x <- 1:5
for (val in x) {
    if (val == 3){
        next
    }
    print(val)
}
```

Output:

```
[1] 1
[1] 2
[1] 4
[1] 5
```

1.6 Return

Return function is used when functions do some processing and the result has to be returned back. This is accomplished with the return() function in R.

Pseudocode:

```
return(expression)
```

Example:

```
mysum = function(a,b)
{
s = a+b
return(s)
}
print(mysum(10,20))
```

Output:

```
[1] 30
```

Summary

Control structures allow you to control the flow of execution of a program. A few common control structures are if-else, for, while, repeat, break, next, and return.

A “for” loop works on an iterable variable and assigns successive values till the end of a sequence. It repeats the same task for changing values of a given variable

Repeat statement iterates a block of code multiple times until a break() statement is encountered, else the loop continues to execute infinitely eventually leading a program to crash.

A Next statement is useful when the current iteration of a loop has to be skipped without terminating it. On encountering next(), the R parser skips further evaluation and starts next iteration of the loop.

Return function is used when functions do some processing and the result has to be returned back.