Loops in R

Dr. Niladri Chakraborty

if-else condition in R.

In R, if-else statements are used to perform conditional execution of code. The basic syntax of an if-else statement in R is as follows:

```
#if (condition) {
  # Code to execute if condition is TRUE
#} else {
  # Code to execute if condition is FALSE
#}
```

Here, condition is a logical expression that evaluates to either TRUE or FALSE. If the condition is TRUE, the code inside the first block (inside the curly braces) is executed; otherwise, the code inside the second block is executed.

```
# Define a variable
x <- 10

# Check if x is greater than 5
if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is less than or equal to 5")
}

## [1] "x is greater than 5"
```

if-else statements can also be nested, allowing for more complex conditional execution of code. Here's an example:

```
# Define a variable
x <- 10

# Check if x is greater than 5
if (x > 5) {
  # If x is greater than 5, check if it's even or odd
  if (x %% 2 == 0) {
    print("x is greater than 5 and even")
  } else {
    print("x is greater than 5 and odd")
  }
} else {
```

```
  print("x is less than or equal to 5")
}

## [1] "x is greater than 5 and even"
```

In this example, we first check if x is greater than 5 using an if statement. If it is, we nest another if-else statement inside to check if x is even or odd. If x is even, we print a message saying that x is greater than 5 and even; otherwise, we print a message saying that x is greater than 5 and odd. If x is less than or equal to 5, we simply print a message saying that x is less than or equal to 5.

For loop

In R, a for loop is a programming construct that allows you to repeat a set of instructions a specified number of times. Here is the basic syntax for a for loop in R:

```
#for (variable in sequence) {
#  statement(s)
#}
```

In this syntax, variable is a loop variable that takes on a different value from the sequence on each iteration of the loop. The statement(s) are the instructions that are executed on each iteration of the loop.

Example 1.

Here is an example of a for loop that prints the numbers 1 to 5:

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

## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Here are some key points to keep in mind when using for loops in R:

The sequence can be any R object that can be converted to a sequence of integers. For example, you can use a vector, a list, or a numeric or integer value.

The loop variable variable is created automatically by the for loop and does not need to be initialized beforehand.

You can use the break statement to exit the for loop prematurely, and the next statement to skip to the next iteration of the loop.

Example 2.

Here is an example of using a for loop to perform addition of a sequence of numbers in R:

```
# Initialize variables
n <- 5
total <- 0

# Loop through the sequence of numbers and add them up
for (i in 1:n) {
  total <- total + i
}

# Print the total
print(total)
```

```
## [1] 15
```

In this example, we initialize two variables n and total. n represents the number of terms we want to add, and total is the sum of the terms.

We then use a for loop to loop through the sequence of numbers from 1 to n. On each iteration of the loop, we add the value of i (the loop variable) to the total variable.

Finally, we print the total variable, which should be the sum of the sequence of numbers from 1 to n. In this case, the output should be 15, which is the sum of the numbers 1 to 5.

Example 3.

```
# Initialize variables
start <- 2
end <- 10
step <- 2
total <- 0

# Loop through the sequence of numbers and add them up
for (i in seq(from=start, to=end, by=step)) {
  print(i)
  total <- total + i
}
```

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

```
# Print the total
print(total)
```

```
## [1] 30
```

Example 4.

Here's an example code for calculating the factorial of a number using a for loop in R:

```
# Define the number to calculate the factorial of
n <- 5

# Initialize the factorial variable to 1
factorial <- 1

# Loop through the numbers from 1 to n and multiply the factorial variable
for (i in 1:n) {
  factorial <- factorial * i
}

# Print the factorial of n
print(factorial)
```

```
## [1] 120
```

While loop

In R, a while loop is a control structure that allows you to repeat a block of code while a specified condition is true. The syntax for a while loop in R is as follows:

```
#while (condition) {
  # code to be executed
#}
```

The condition is a logical expression that is evaluated at the beginning of each iteration of the loop. If the condition is true, the code within the curly braces is executed. This process is repeated until the condition is false.

Example 1.

```
# Define the number to calculate the factorial of
n <- 5

# Initialize the factorial variable to 1
factorial <- 1

# Initialize the counter variable i to 1
i <- 1

# Use a while loop to calculate the factorial of n
while (i <= n) {
  factorial <- factorial * i
  i <- i + 1
}
```

```r
# Print the factorial of n
print(factorial)

## [1] 120
```

In this example, we define the number we want to calculate the factorial of as n. We then initialize the factorial variable to 1, since the factorial of 0 and 1 are both 1. We also initialize the counter variable i to 1.

We then use a while loop to repeatedly multiply the factorial variable by i and increment i by 1 until i is greater than n.

Finally, we print the factorial of n. In this case, the output should be 120, which is the factorial of 5 (5 * 4 * 3 * 2 * 1).

While loops are useful when you want to repeat a block of code a variable number of times, as the number of iterations depends on the value of a variable or the result of a logical expression. However, care should be taken when using while loops to avoid infinite loops, where the condition is always true and the loop never terminates.

Here's an example of how you can use a while loop in R to avoid infinite loops by ensuring that the loop condition eventually becomes false:

Example 2.

```r
# Initialize the variable i to 1
i <- 1

# Use a while loop to print the first 10 even numbers
while (i <= 20) {
  if (i %% 2 == 0) {
    print(i)
  }
  i <- i + 1
}

## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
## [1] 12
## [1] 14
## [1] 16
## [1] 18
## [1] 20
```

n this example, we initialize the variable i to 1 and use a while loop to print the first 10 even numbers. Inside the loop, we check if i is divisible by 2 (using the modulus operator %%) and if so, we print it to the console.

Notice that we set the loop condition to i <= 20 instead of TRUE or some other constant value. This ensures that the loop will eventually terminate once i becomes greater than 20.

If we had set the loop condition to TRUE, the loop would have continued infinitely, printing even numbers forever and never terminating. This is an example of an infinite loop, which can be problematic if left unchecked.

An example when while loop is more appropriate than for loop.

In general, while loops and for loops are interchangeable in most cases in R. However, there are certain situations where while loops may be more appropriate than for loops. One example is when the number of iterations is not known in advance, and depends on a condition that may change within the loop.

```
# Define a vector of numbers
numbers <- rpois(100,1)

# Initialize the sum and counter variables
sum <- 0
i <- 1

# Use a while loop to calculate the sum of the numbers
while (sum < 50 && i <= length(numbers)) {
  sum <- sum + numbers[i]
  i <- i + 1
}

# Print the sum of the numbers
print(sum)

## [1] 50
```

In this example, we have a vector of numbers and want to calculate the sum of the numbers until the sum is greater than or equal to 50. We also want to ensure that we don't exceed the length of the vector while calculating the sum.

A while loop is more appropriate than a for loop in this case, as we don't know in advance how many iterations we will need to reach the desired sum. Instead, we use a while loop to repeat the calculation until the sum meets our condition or we reach the end of the vector.

If we had used a for loop instead, we would need to know the exact number of iterations in advance, which is not possible in this case.

Repeat loop

In R, repeat loops are used to repeatedly execute a block of code until a certain condition is met. The basic syntax of a repeat loop in R is as follows:

```
#repeat {
  # Code to execute
#  if (condition) {
#    break
#  }
#}
```

Here, condition is a logical expression that is used to test if the loop should continue or not. If the condition is TRUE, the loop continues to execute the code inside the curly braces. If the condition is FALSE, the loop terminates.

Unlike for and while loops, repeat loops do not have a specific iteration count or termination condition. Instead, they rely on the programmer to provide a condition inside the loop that will eventually cause it to terminate. This makes repeat loops useful for situations where the termination condition is not known in advance, or where the loop needs to continue until a certain condition is met.

Example 1.

```
repeat {
  # Generate a random number between 1 and 10
  x <- sample(1:10, 1)

  # Check if the number is greater than 7
  if (x > 7) {
    # If it is, break out of the loop
    break
  }
}

# Print the final value of x
print(x)
```

```
## [1] 10
```

In this example, we use a repeat loop to generate a random number between 1 and 10 using the sample() function. We then check if the number is greater than 7 using an if statement. If it is, we break out of the loop using the break statement. If the number is less than or equal to 7, the loop continues to execute, generating another random number and checking it again.

Note that repeat loops can potentially run forever if the termination condition is never met. To avoid this, it's important to make sure that the termination condition will eventually be met.

Example 2.

Here's an example where a repeat loop might be more appropriate than a for or while loop in a statistical context:

Suppose you are trying to estimate the probability of a rare event that occurs with a very low probability, say 0.001. You want to estimate the probability by simulating the event a large number of times, say 10,000, and counting how many times the event occurs. However, the event is so rare that it might take a long time to simulate 10,000 occurrences of the event.

```r
# Set the desired number of occurrences
num_occurrences <- 10000

# Initialize a counter variable
occurrence_count <- 0

# Start a repeat loop
repeat {
  # Simulate the event
  if (runif(1) < 0.001) {
    # If the event occurs, increment the counter variable
    occurrence_count <- occurrence_count + 1
  }

  # Check if we have reached the desired number of occurrences
  if (occurrence_count >= num_occurrences) {
    # If we have reached the desired number of occurrences, break out of the
loop
    break
  }
}

# Calculate the estimated probability of the event
est_prob <- occurrence_count / num_occurrences

# Print the estimated probability
cat("Estimated probability: ", est_prob, "\n")

## Estimated probability:  1
```

The cat() function in R is used to concatenate and print one or more objects to the console or to a file. It stands for "concatenate and print", and its main purpose is to output one or more R objects as a continuous string of text, separated by spaces or other specified characters.

In this example, we start a repeat loop that will continue until we have simulated the event the desired number of times. Inside the loop, we simulate the event using the runif() function, which generates a random number between 0 and 1. If the random number is less than the probability of the event (0.001 in this case), we count the occurrence by incrementing the occurrence_count variable. We then check if we have reached the desired number of occurrences using the >= operator. If we have reached the desired number of occurrences, we break out of the loop using the break statement.

More advanced 'loop' functions.

In R, apply, lapply, and sapply are functions that are used to apply a given function to a collection of elements, such as a matrix or list. Here's a brief explanation of each function:

- apply(): This function is used to apply a function to the rows or columns of a matrix or array. The function can be any R function that takes a vector of values as input and returns a single value as output. The output of apply() is a vector or array of the results of applying the function to each row or column.
- lapply(): This function is used to apply a function to each element of a list. The function can be any R function that takes a vector of values as input and returns a single value as output. The output of lapply() is a list of the results of applying the function to each element of the original list.
- sapply(): This function is similar to lapply(), but attempts to simplify the output into a vector or matrix if possible. If the output of lapply() is a list where each element has the same length, sapply() will attempt to simplify the output into a matrix or vector.

Example 1.

```
# Apply function to each row of a matrix
m <- matrix(1:12, nrow = 4)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
apply(m, 1, sum)  # Output: 15 18 21 24
```

```
## [1] 15 18 21 24
```

```
# Apply function to each column of a matrix
m <- matrix(1:12, nrow = 4)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
apply(m, 1, sum)  # Output: 15 18 21 24
```

```
## [1] 15 18 21 24
```

```r
# Apply function to each element of a list
l <- list(a = c(1, 2, 3), b = c(4, 5, 6))
lapply(l, mean)   # Output: $a 2 $b 5
```

```
## $a
## [1] 2
##
## $b
## [1] 5
```

```r
# Simplify output using sapply
l <- list(a = c(1, 2, 3), b = c(4, 5, 6))
sapply(l, mean)   # Output: a b 2 5
```

```
## a b
## 2 5
```

Example 2.

use apply() function in R to calculate the column means of a matrix.

```r
# Create a matrix
m <- matrix(1:1000, nrow = 10)
m[,1:5]
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1   11   21   31   41
##  [2,]    2   12   22   32   42
##  [3,]    3   13   23   33   43
##  [4,]    4   14   24   34   44
##  [5,]    5   15   25   35   45
##  [6,]    6   16   26   36   46
##  [7,]    7   17   27   37   47
##  [8,]    8   18   28   38   48
##  [9,]    9   19   29   39   49
## [10,]   10   20   30   40   50
```

```r
# Calculate the column means using apply()
col_means <- apply(m, 2, mean)

# Print the column means
col_means
```

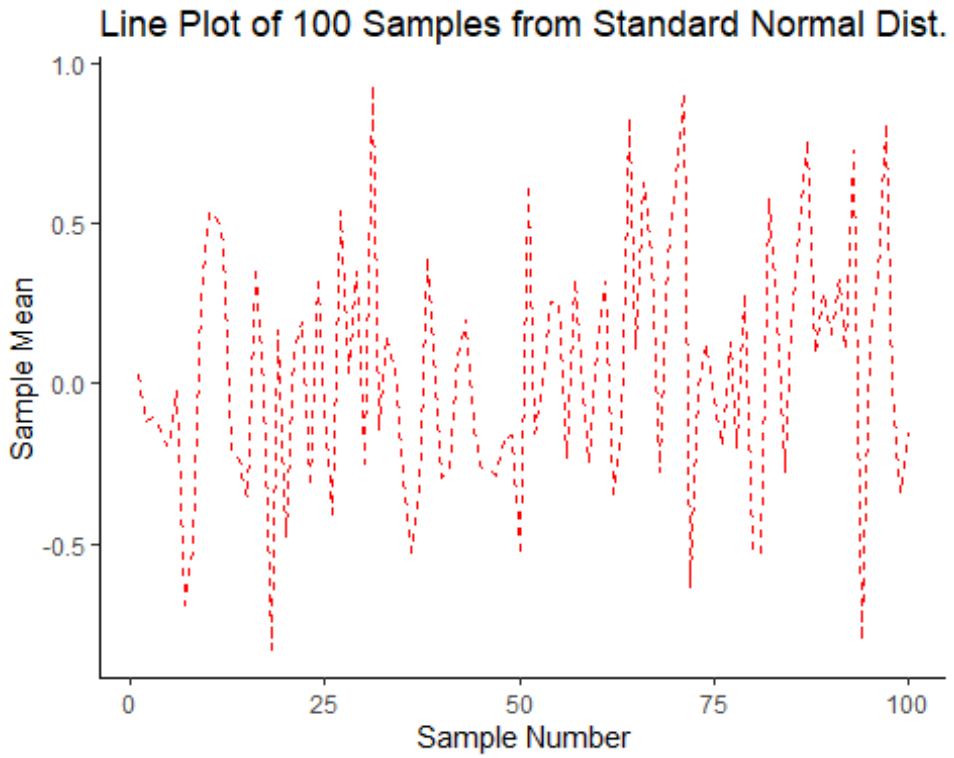
```
##   [1]   5.5  15.5  25.5  35.5  45.5  55.5  65.5  75.5  85.5  95.5 105.5
115.5
##  [13] 125.5 135.5 145.5 155.5 165.5 175.5 185.5 195.5 205.5 215.5 225.5
235.5
##  [25] 245.5 255.5 265.5 275.5 285.5 295.5 305.5 315.5 325.5 335.5 345.5
355.5
##  [37] 365.5 375.5 385.5 395.5 405.5 415.5 425.5 435.5 445.5 455.5 465.5
475.5
##  [49] 485.5 495.5 505.5 515.5 525.5 535.5 545.5 555.5 565.5 575.5 585.5
```

```
595.5
##  [61] 605.5 615.5 625.5 635.5 645.5 655.5 665.5 675.5 685.5 695.5 705.5
715.5
##  [73] 725.5 735.5 745.5 755.5 765.5 775.5 785.5 795.5 805.5 815.5 825.5
835.5
##  [85] 845.5 855.5 865.5 875.5 885.5 895.5 905.5 915.5 925.5 935.5 945.5
955.5
##  [97] 965.5 975.5 985.5 995.5
```

A more advanced example.

Here's an example code in R to draw 100 random samples of size 5 from a standard normal distribution, calculate the mean for each sample, and create a line plot:

```r
library(ggplot2)
library(hrbrthemes)

## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use
these themes.

##       Please use hrbrthemes::import_roboto_condensed() to install Roboto
Condensed and

##       if Arial Narrow is not on your system, please see
https://bit.ly/arialnarrow

# Set the seed for reproducibility
set.seed(123)

# Draw 100 random samples of size 5 from a standard normal distribution
samples <- matrix(rnorm(100*5), ncol=5)

# Calculate the mean for each sample
sample_means <- apply(samples, 1, mean)

# Create a data frame with the sample means
df <- data.frame(sample_means)

# Create a line plot of the sample means using ggplot
ggplot(df, aes(x=1:nrow(df), y=sample_means)) +
  geom_line(linetype="dashed", color="red") +
  xlab("Sample Number") +
  ylab("Sample Mean") +
  ggtitle("Line Plot of 100 Samples from Standard Normal Dist.")+
  theme_classic()
```


Line Plot of 100 Samples from Standard Normal Dist.
Sample Mean
1.0
0.5
0.0
-0.5
0
25
50
75
100
Sample Number