R for Loop

A `for` loop is used to iterate over a list, vector or any other object of elements. The syntax of `for` loop is:

```
for (value in sequence) {
  # block of code
}
```

Here, `sequence` is an object of elements and `value` takes in each of those elements. In each iteration, the block of code is executed. For example,

```
numbers = c(1, 2, 3, 4, 5)

# for loop to print all elements in numbers
for (x in numbers) {
  print(x)
}
```

Output

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

In this program, we have used a `for` loop to iterate through a sequence of numbers called `numbers`. In each iteration, the variable `x` stores the element from the sequence and the block of code is executed.

Example 1: Count the Number of Even Numbers

Let's use a `for` loop to count the number of even numbers stored inside a vector of numbers.

```
# vector of numbers
num = c(2, 3, 12, 14, 5, 19, 23, 64)
```

```
# variable to store the count of even numbers
count = 0

# for loop to count even numbers
for (i in num) {

  # check if i is even
  if (i %% 2 == 0) {
    count = count + 1
  }
}


print(count)
```

Output

```
[1] 4
```

In this program, we have used a `for` loop to count the number of even numbers in the `num` vector. Here is how this program works:

- We first initialized the `count` variable to **0**. We use this variable to store the count of even numbers in the `num` vector.
- We then use a `for` loop to iterate through the `num` vector using the variable `i`.

```
- for (i in num) {
-   # code block
}
```

- Inside the `for` loop, we check if each element is divisible by **2** or not. If yes, then we increment `count` by **1**.

```
- if (i %% 2 == 0) {
```

```
    count = count + 1
}
```

Example 2: for Loop With break Statement

You can use the `break` statement to exit from the `for` loop in any iteration.

For example,

```
# vector of numbers
numbers = c(2, 3, 12, 14, 5, 19, 23, 64)

# for loop with break
for (i in numbers) {

  # break the loop if number is 5
  if( i == 5) {
    break
  }

  print(i)
}
```

Output

```
[1] 2
[1] 3
[1] 12
[1] 14
```

Here, we have used an `if` statement inside the `for` loop. If the current element is equal to `5`, we break the loop using the `break` statement. After this, no iteration will be executed.

Example 3: for Loop With next Statement

Instead of terminating the loop, you can skip an iteration using the `next` statement. For example,

```
# vector of numbers
numbers = c(2, 3, 12, 14, 5, 19, 23, 64)

# for loop with next
for (i in numbers) {

  # use next to skip odd numbers
  if( i %% 2 != 0) {
    next
  }

  print(i)
}
```

Output

```
[1] 2
[1] 12
[1] 14
[1] 64
```

Here, we have used an `if` statement inside the `for` loop to check for odd numbers. If the number is odd, we skip the iteration using the `next` statement and print only even numbers.

Nested for Loops

You can include a `for` loop inside another `for` loop to create a nested loop. Consider the example below. Suppose we have two sequences of numbers. We want to print all the combinations where the sum of numbers in both the sequences is even.

```
# vector of numbers
sequence_1 = c(1, 2, 3)
sequence_2 = c(1, 2, 3)

# nested for loop
for (i in sequence_1) {
  for (j in sequence_2) {

    # check if sum is even
    if ( (i+j) %% 2 == 0 ) {
      print(paste(i, j))
    }

  }
}
```

Output

```
[1] "1 1"
[1] "1 3"
[1] "2 2"
[1] "3 1"
[1] "3 3"
```

In the above program, we have created two sequences: `sequence_1` and `sequence_2`, both containing numbers from `1` to `3`. We then used a nested `for` loop to iterate through the sequences. The outer loop iterates through `sequence_1` and the inner loop iterates through `sequence_2`.

```
for (i in sequence_1) {
  for (j in sequence_2) {
    # code block
  }
}
```

In each iteration,

- `i` stores the current number of `sequence_1`
- `j` stores the current number of `sequence_2`

  The `if` statement inside the nested loops checks if `i + j` is even or not. If it is, then we print `i` and `j`.

```
if ( (i+j) %% 2 == 0 ) {
  print(paste(i, j))
}
```

R break and next

In this tutorial, you'll learn about the break and next statements in R with the help of examples.

We use the R `break` and `next` statements to alter the flow of a program. These are also known as jump statements in programming:

- `break` - terminate a looping statement
- `next` - skips an iteration of the loop

R break Statement

You can use a `break` statement inside a loop (`for`, `while`, `repeat`) to terminate the execution of the loop. This will stop any further iterations.
The syntax of the `break` statement is:

```
if (test_expression) {
  break
}
```

The `break` statement is often used inside a conditional (`if...else`) statement in a loop. If the condition inside the `test_expression` returns `True`, then the `break` statement is executed. For example,

```
# vector to be iterated over
x = c(1, 2, 3, 4, 5, 6, 7)

# for loop with break statement
for(i in x) {
```

```
    # if condition with break
    if(i == 4) {
        break
    }

    print(i)
}
```

Output

```
[1] 1
[1] 2
[1] 3
```

Here, we have defined a vector of numbers from `1` to `7`. Inside the `for` loop, we check if the current number is `4` using an `if` statement.
If yes, then the `break` statement is executed and no further iterations are carried out. Hence, only numbers from `1` to `3` are printed.

---

break Statement in Nested Loop

If you have a nested loop and the `break` statement is inside the inner loop, then the execution of only the inner loop will be terminated.
Let's check out a program to use break statements in a nested loop.

```
# vector to be iterated over
x = c(1, 2, 3)
y = c(1, 2, 3)

# nested for loop with break statement
for(i in x) {
    for (j in y) {
        if (i == 2 & j == 2) {
            break
        }
        print(paste(i, j))
    }
```

```
}
```

Output

```
[1] "1 1"
[1] "1 2"
[1] "1 3"
[1] "2 1"
[1] "3 1"
[1] "3 2"
[1] "3 3"
```

Here, we have a `break` statement inside the inner loop.

We have used it inside a conditional statement such that if both the numbers are equal to `2`, the inner loop gets terminated.

The flow then moves to the outer loop. Hence, the combination `(2, 2)` is never printed.

R next Statement

In R, the `next` statement skips the current iteration of the loop and starts the loop from the next iteration.

The syntax of the `next` statement is:

```
if (test_condition) {
  next
}
```

If the program encounters the `next` statement, any further execution of code from the current iteration is skipped, and the next iteration begins.

Let's check out a program to print only even numbers from a vector of numbers.

```
# vector to be iterated over
x = c(1, 2, 3, 4, 5, 6, 7, 8)
```

```
# for loop with next statement
for(i in x) {

    # if condition with next
    if(i %% 2 != 0) {
        next
    }

    print(i)
}
```

Output

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

Here, we have used an `if` statement to check whether the current number in the loop is odd or not.
If yes, the `next` statement inside the if block is executed, and the current iteration is skipped.

R Vectors

In this tutorial, you will learn about vectors in R with the help of examples.

A vector is the basic data structure in R that stores data of similar types. For example,

Suppose we need to record the age of **5** employees. Instead of creating **5** separate variables, we can simply create a vector.

Elements of a Vector

Create a Vector in R

In R, we use the `c()` function to create a vector. For example,

```
# create vector of string types
employees <- c("Sabby", "Cathy", "Lucy")

print(employees)

# Output: [1] "Sabby" "Cathy" "Lucy"
```

In the above example, we have created a vector named `employees` with elements: `Sabby`, `Cathy`, and `Lucy`.
Here, the `c()` function creates a vector by combining three different elements of `employees` together.

Access Vector Elements in R

In R, each element in a vector is associated with a number. The number is known as a vector index.

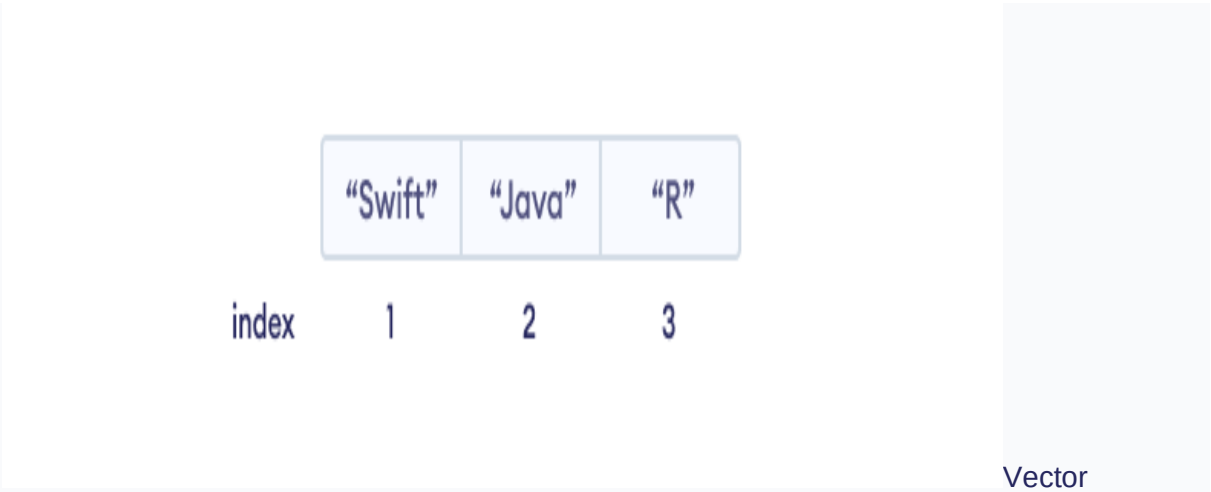
We can access elements of a vector using the index number **(1, 2, 3 ...)**.
For example,

```
# a vector of string type
languages <- c("Swift", "Java", "R")

# access first element of languages
print(languages[1])  # "Swift"

# access third element of languages
print(languages[3]). # "R"
```

In the above example, we have created a vector named `languages`. Each element of the vector is associated with an integer number.



Vector

Indexing in R

Here, we have used the vector index to access the vector elements

- `languages[1]` - access the first element `"Swift"`
- `languages[3]` - accesses the third element `"R"`

> **Note**: In R, the vector index always starts with **1**. Hence, the first element of a vector is present at index **1**, second element at index **2** and so on.

Modify Vector Element

To change a vector element, we can simply reassign a new value to the specific index. For example,

```
dailyActivities <- c("Eat","Repeat")
cat("Initial Vector:", dailyActivities)

# change element at index 2
dailyActivities[2] <- "Sleep"

cat("\nUpdated Vector:", dailyActivities)
```

Output

```
Initial Vector: Eat Repeat
```

```
Updated Vector: Eat Sleep
```

Here, we have changed the vector element at index **2** from `"Repeat"` to `"Sleep"` by simply assigning a new value.

Numeric Vector in R

Similar to strings, we use the `c()` function to create a numeric vector. For example,

```
# a vector with number sequence from 1 to 5
numbers <- c(1, 2, 3, 4, 5)

print(numbers)

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

Here, we have used the `C()` function to create a vector of numeric sequence called `numbers`.

However, there is an efficient way to create a numeric sequence. We can use the `:` operator instead of `C()`.

Create a Sequence of Number in R

In R, we use the `:` operator to create a vector with numerical values in sequence. For example,

```
# a vector with number sequence from 1 to 5
numbers <- 1:5

print(numbers)
```

Output

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

Here, we have used the `:` operator to create the vector named `numbers` with numerical values in sequence i.e. **1** to **5.**

Repeat Vectors in R

In R, we use the `rep()` function to repeat elements of vectors. For example,

```
# repeat sequence of vector 2 times
numbers <- rep(c(2,4,6), times = 2)

cat("Using times argument:", numbers)
```

Output

```
Using times argument: 2 4 6 2 4 6
```

In the above example, we have created a numeric vector with elements **2, 4, 6**. Notice the code,

```
rep(numbers, times=2)
```

Here,

- `numbers` - vector whose elements to be repeated
- `times = 2` - repeat the vector two times

We can see that we have repeated the whole vector two times. However, we can also repeat each element of the vector. For this we use the `each` parameter.

Let's see an example.

```
# repeat each element of vector 2 times
numbers <- rep(c(2,4,6), each = 2)

cat("\nUsing each argument:", numbers)
```

Output

```
Using each argument: 2 2 4 4 6 6
```

In the above example, we have created a numeric vector with elements **2, 4, 6**. Notice the code,

```
rep(numbers, each = 2)
```

Here, `each = 2` - repeats each element of vector two times

Loop Over a R Vector

We can also access all elements of the vector by using a for loop. For example,

In R, we can also loop through each element of the vector using the for loop. For example,

```
numbers <- c(1, 2, 3, 4, 5)

# iterate through each elements of numbers
for (number in numbers) {
    print(number)
}
```

Output

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

Length of Vector in R

We can use the `length()` function to find the number of elements present inside the vector. For example,

```
languages <- c("R", "Swift", "Python", "Java")

# find total elements in languages using length()
cat("Total Elements:", length(languages))
```

Output

```
Total Elements: 4
```

Here, we have used `length()` to find the length of the `languages` vector.

R List

In this tutorial, we will learn about list in R with the help of examples.

A List is a collection of similar or different types of data.

In R, we use the `list()` function to create a list. For example,

```
# list with similar type of data
list1 <- list(24, 29, 32, 34)

# list with different type of data
list2 <- list("Ranjy", 38, TRUE)
```

Here,

- `list1` - list of integers
- `list2` - list containing string, integer, and boolean value

Access List Elements in R

In R, each element in a list is associated with a number. The number is known as a list index.

We can access elements of a list using the index number **(1, 2, 3 …)**. For example,

```
list1 <- list(24, "Sabby", 5.4, "Nepal")

# access 1st item
print(list1[1]) # 24

# access 4th item
print(list1[4]) # Nepal
```

In the above example, we have created a list named `list1`.
Here, we have used the vector index to access the vector elements

- `list1[1]` - access the first element **24**
- `list1[4]` - accesses the third element `"Nepal"`

> **Note**: In R, the list index always starts with **1**. Hence, the first element of a list is present at index **1**, second element at index **2** and so on.

Modify a List Element in R

To change a list element, we can simply reassign a new value to the specific index. For example,

```
list1 <- list(24, "Sabby", 5.4, "Nepal")

# change element at index 2
list1[2] <- "Cathy"

# print updated list
print(list1)
```

Output

```
[[1]]
[1] 24
```

```
[[2]]
[1] "Cathy"

[[3]]
[1] 5.4

[[4]]
[1] "Nepal"
```

Here, we have reassigned a new value to index **2** to change the list element from `"Sabby"` to `"Cathy"`.

Add Items to R List

We use the `append()` function to add an item at the end of the list. For example,

```
list1 <- list(24, "Sabby", 5.4, "Nepal")

# using append() function
append(list1, 3.14)
```

Output

```
[[1]]
[1] 24

[[2]]
[1] "Sabby"

[[3]]
[1] 5.4

[[4]]
[1] "Nepal"

[[5]]
[1] 3.14
```

In the above example, we have created a list named `list1`. Notice the line,

```
append(list1, 3.14)
```

Here, `append()` adds **3.14** at the end of the list.

Remove Items From a List in R

R allows us to remove items for a list. We first access elements using a list index and add negative sign `-` to indicate we want to delete the item. For example,

- `[-1]` - removes 1st item
- `[-2]` - removes 2nd item and so on.

```
list1 <- list(24, "Sabby", 5.4, "Nepal")

# remove 4th item
print(list1[-4]) # Nepal
```

Output

```
[[1]]
[1] "Sabby"

[[2]]
[1] 5.4

[[3]]
[1] "Nepal"
```

Here, `list[-4]` removes the 4th item of `list1`.

Length of R List

In R, we can use the `length()` function to find the number of elements present inside the list. For example,

```
list1 <- list(24, "Sabby", 5.4, "Nepal")

# find total elements in list1 using length()
cat("Total Elements:", length(list1))
```

Output

```
Total Elements: 4
```

Here, we have used the `length()` function to find the length of `list1`. Since there are **4** elements in `list1` so `length()` returns **4**.

Loop Over a List

In R, we can also loop through each element of the list using the for loop. For example,

```
items <- list(24, "Sabby", 5.4, "Nepal")

# iterate through each elements of numbers
for (item in items) {
    print(item)
}
```

Output

```
[1] 24
[1] "Sabby"
[1] 5.4
[1] "Nepal"
```
