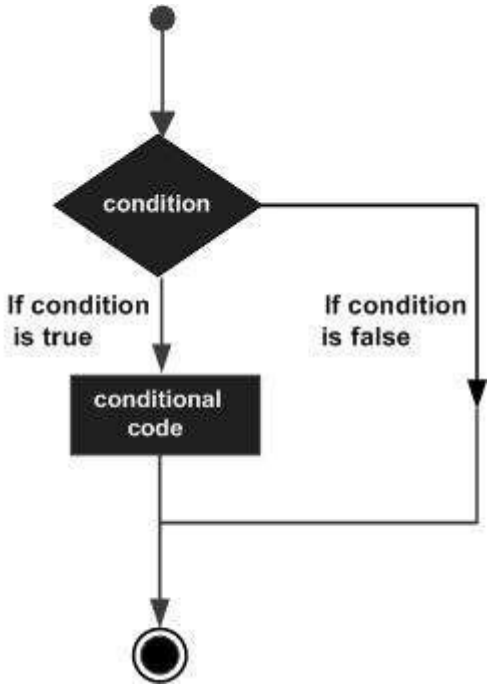
R - Decision making

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be **true**, and optionally, other statements to be executed if the condition is determined to be **false**.

Following is the general form of a typical decision making structure found in most of the programming languages –



R provides the following types of decision making statements. Click the following links to check their detail.

Sr.No.	Statement & Description
1	**if statement**

	An **if** statement consists of a Boolean expression followed by one or more statements.
2	**if...else statement** An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.
3	**switch statement** A **switch** statement allows a variable to be tested for equality against a list of values.

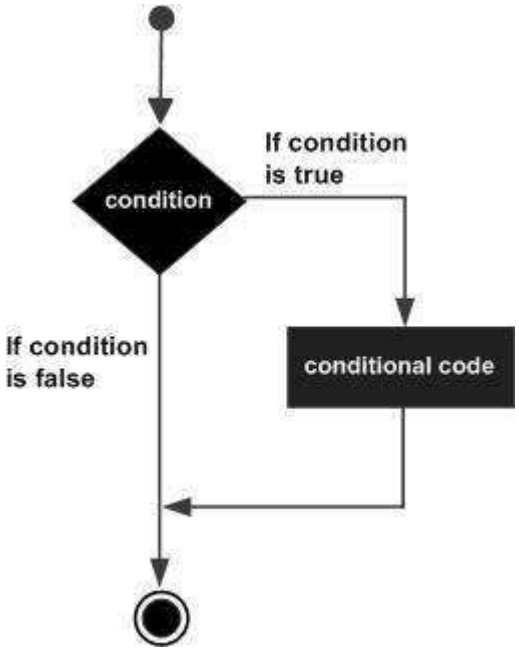
R - If Statement

An if statement consists of a Boolean expression followed by one or more statements.
Syntax
The basic syntax for creating an if statement in R is:

```
if(boolean_expression) {
// statement(s) will execute if the boolean expression is true.
}
```

If the Boolean expression evaluates to be true, then the block of code inside the if statement will be executed. If Boolean expression evaluates to be false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flow Diagram



Example

```
x <- 30L
if(is.integer(x)){
print("X is an Integer")
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "X is an Integer"
```

```
> x <- 30L
> if(is.integer(x)){
+     print("X is an Integer")
+ } else{}
[1] "X is an Integer"
```

R – If...Else Statement
An if statement can be followed by an optional else statement which executes when the boolean expression is false.
Syntax

The basic syntax for creating an if...else statement in R is:

```
if(boolean_expression) {
// statement(s) will execute if the boolean expression is true.
} else {
// statement(s) will execute if the boolean expression is false.
}
```

```
> x <- 30
> if(is.integer(x)){
+     print("X is an Integer")
+ } else{ print("x is not an integer")}
[1] "x is not an integer"
```

If the Boolean expression evaluates to be true, then the if block of code will be executed, otherwise else block of code will be executed.
Flow Diagram
Example

```
x <- c("what","is","truth")
if("Truth" %in% x){
print("Truth is found")
} else {
print("Truth is not found")
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "Truth is not found"
```

Here "Truth" and "truth" are two different strings.

The if...else if...else Statement

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.
When using if, else if, else statements there are few points to keep in mind.

- An if can have zero or one else and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax
The basic syntax for creating an if...else if...else statement in R is:

```
if(boolean_expression 1) {
// Executes when the boolean expression 1 is true.
}else if( boolean_expression 2) {
// Executes when the boolean expression 2 is true.
}else if( boolean_expression 3) {
// Executes when the boolean expression 3 is true.
}else {
// executes when none of the above condition is true.
}
```

Example

```
x <- c("what","is","truth")
if("Truth" %in% x){
print("Truth is found the first time")
} else if ("truth" %in% x) {
print("truth is found the second time")
} else {
print("No truth found")
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "truth is found the second time"
```

```
> x <- c("what","is","truth")
> if("Truth" %in% x){
+     print("Truth is found the first time")
+ } else if ("truth" %in% x) {
+     print("truth is found the second time")
+ } else {
+     print("No truth found")
+ }
[1] "truth is found the second time"
```

R – Switch Statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax

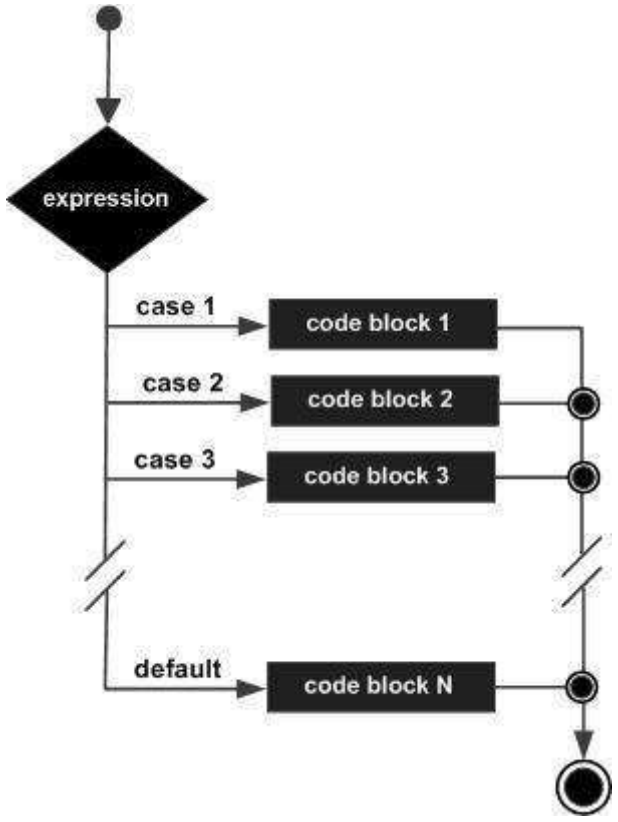
The basic syntax for creating a switch statement in R is :

```
switch(expression, case1, case2, case3....)
```

The following rules apply to a switch statement:

- If the value of expression is not a character string it is coerced to integer.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- If the value of the integer is between 1 and nargs()-1 (The max number of arguments)then the corresponding element of case condition is evaluated and the result returned.
- If expression evaluates to a character string then that string is matched (exactly) to the names of the elements.
- If there is more than one match, the first matching element is returned.
- No Default argument is available.
- In the case of no match, if there is a unnamed element of ... its value is returned.
- (If there is more than one such argument an error is returned.)

Flow Diagram



Example

```
x <- switch(
3,
"first",
"second",
"third",
```

```
"fourth"
)
print(x)
```

When the above code is compiled and executed, it produces the following result:

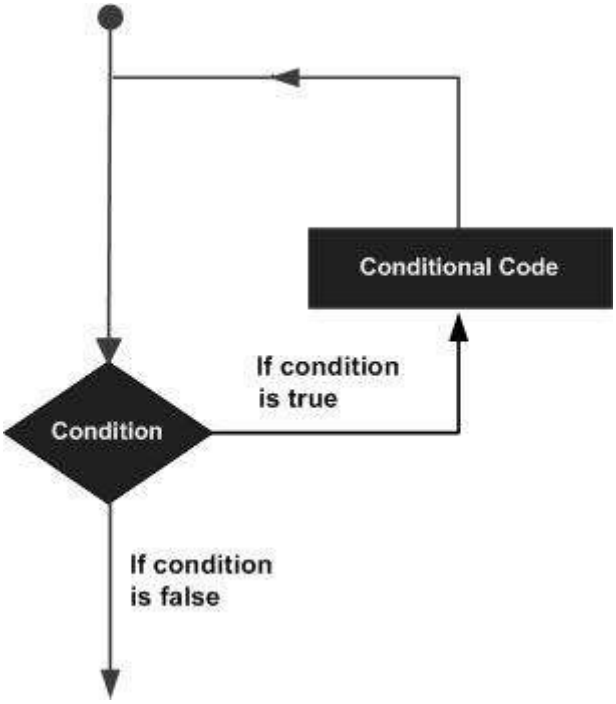
```
[1] "third"
```

R - Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and the following is the general form of a loop statement in most of the programming languages –



R programming language provides the following kinds of loop to handle looping requirements.

Sr.No.	Loop Type & Description
1	**repeat loop** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	**while loop** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	**for loop**

	Like a while statement, except that it tests the condition at the end of the loop body.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

R supports the following control statements. Click the following links to check their detail.

Sr.No.	Control Statement & Description
1	**break statement** Terminates the **loop** statement and transfers execution to the statement immediately following the loop.
2	**Next statement** The **next** statement simulates the behavior of R switch.

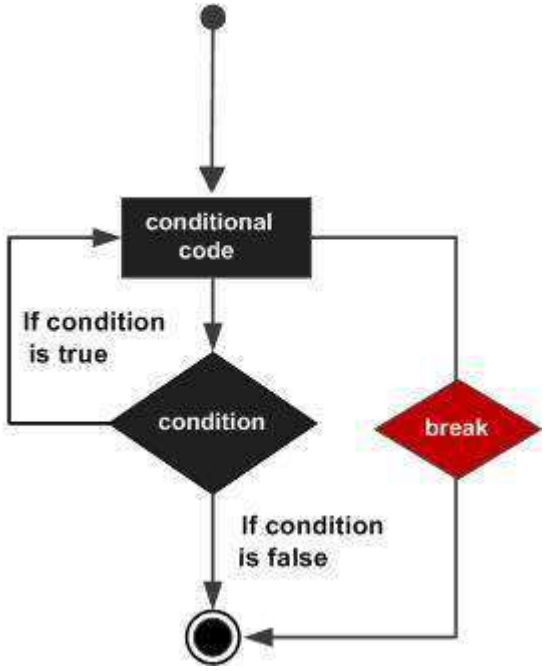
R - Repeat Loop

The Repeat loop executes the same code again and again until a stop condition is met.
Syntax
The basic syntax for creating a repeat loop in R is:

```
repeat {
commands
```

```
if(condition){
break
}
}
```

Flow Diagram



Example

```
v <- c("Hello","loop")
cnt <- 2
repeat{
print(v)
cnt <- cnt+1
if(cnt > 5){
break
}
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
```

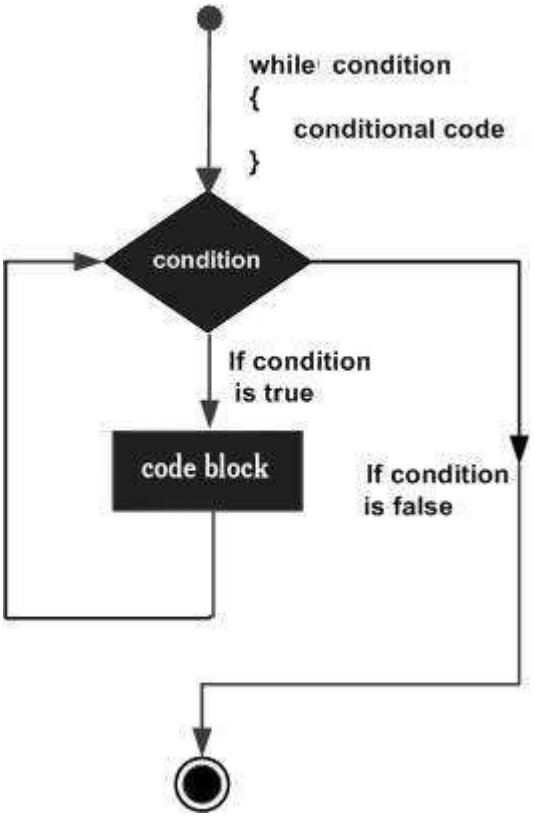
```
> v <- c("Hello","loop")
> cnt <- 2
> repeat{
+     print(v)
+     cnt <- cnt+1
+     if(cnt > 5){
+         break
+     }
+ }
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
>
```

R - While Loop

The While loop executes the same code again and again until a stop condition is met.
Syntax
The basic syntax for creating a while loop in R is :

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

Flow Diagram



Here key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.
Example

```
v <- c("Hello","while loop")
cnt <- 2
while (cnt < 7){
print(v)
cnt = cnt + 1
}
```

When the above code is compiled and executed, it produces the following result :

```
[1] "Hello" "while loop"
[1] "Hello" "while loop"
[1] "Hello" "while loop"
[1] "Hello" "while loop"
[1] "Hello" "while loop"
```

```
> v <- c("Hello","while loop")
> cnt <- 2
> while (cnt < 7){
+     print(v)
+     cnt = cnt + 1
+ }
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
>
```

R – For Loop

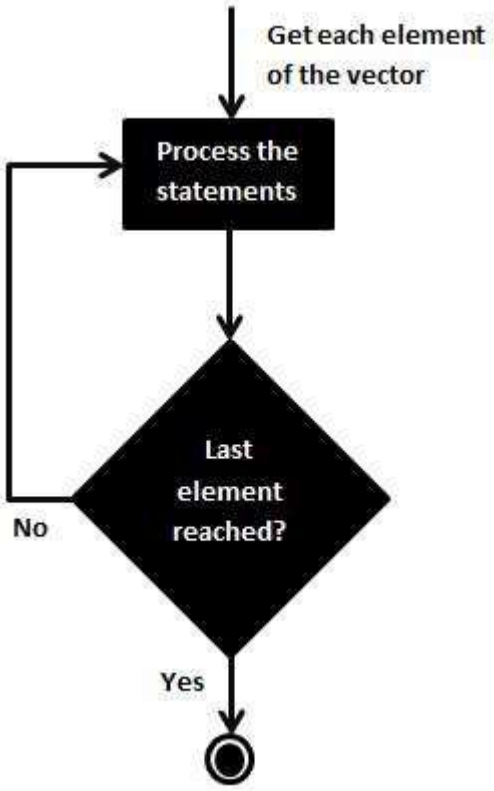
A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The basic syntax for creating a for loop statement in R is:

```
for (value in vector) {
statements
}
```

Flow Diagram



R's for loops are particularly flexible in that they are not limited to integers, or even numbers in the input. We can pass character vectors, logical vectors, lists or expressions.

Example

```
v <- LETTERS[1:4]
for ( i in v) {
print(i)
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "A"
[1] "B"
[1] "C"
[1] "D"
```

```
> v <- LETTERS[1:4]
> for ( i in v) {
+     print(i)
+ }
[1] "A"
[1] "B"
[1] "C"
[1] "D"
>
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
R supports the following control statements. Click the following links to check their detail.

R – Break Statement

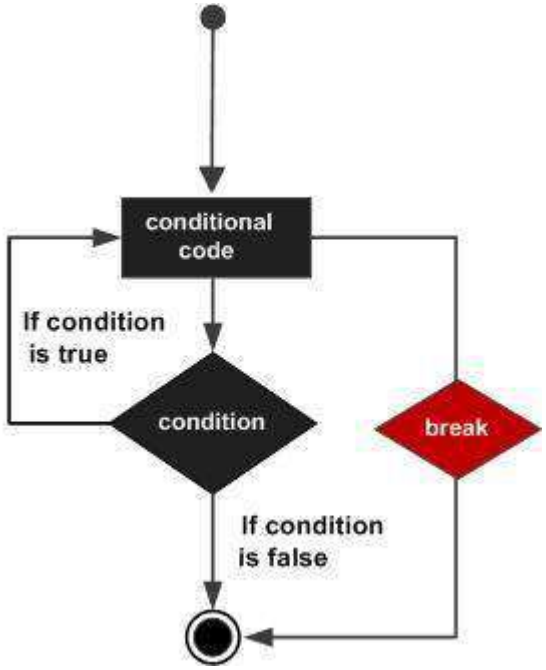
The break statement in R programming language has the following two usages:

- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement (covered in the next chapter).

Syntax
The basic syntax for creating a break statement in R is:

```
break
```

Flow Diagram



Examples

```
v <- c("Hello","loop")
cnt <- 2
repeat{
print(v)
cnt <- cnt+1
if(cnt > 5){
break
}
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
```

```
> v <- c("Hello","loop")
> cnt <- 2
> repeat{
+     print(v)
+     cnt <- cnt+1
+     if(cnt > 5){
+         break
+     }
+ }
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
>
```

R – Next Statement

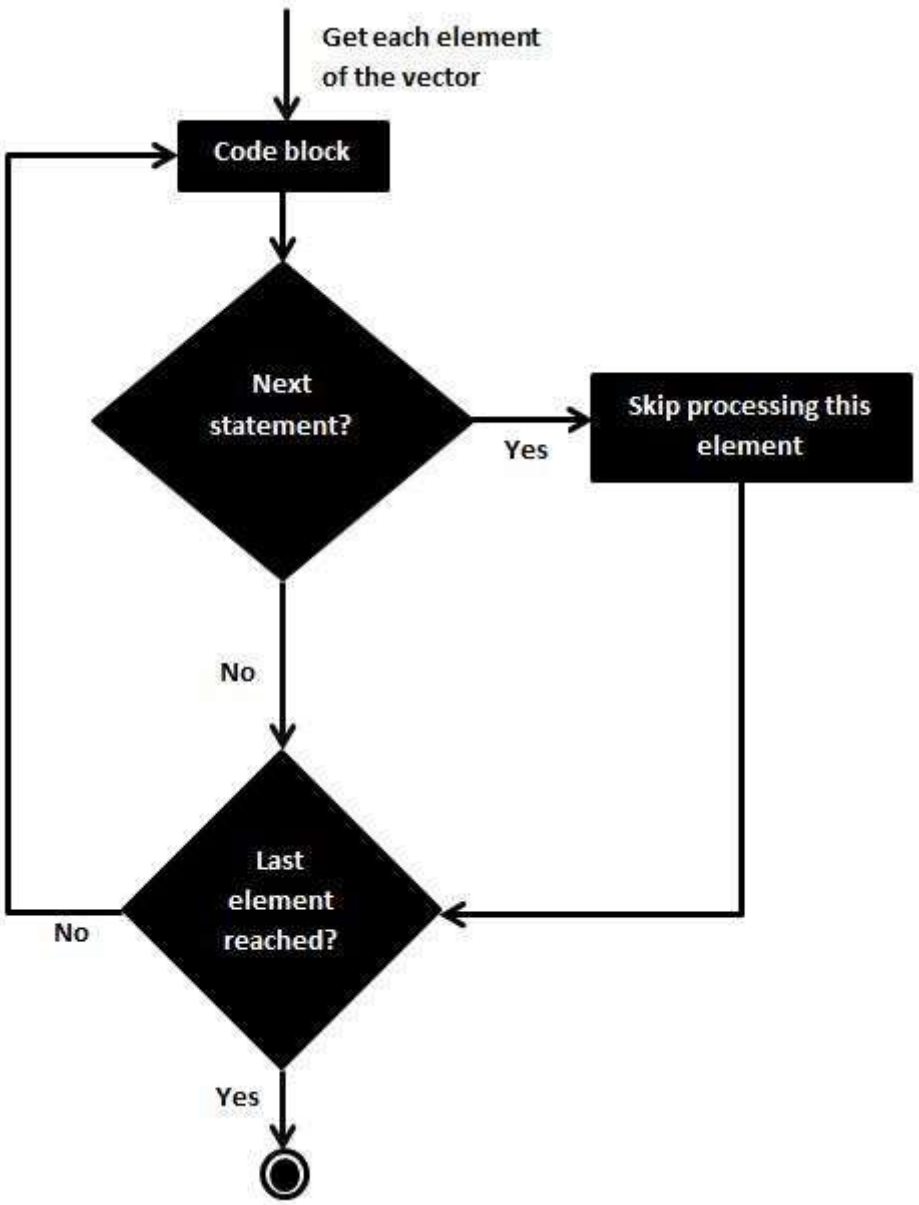
The next statement in R programming language is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R parser skips further evaluation and starts next iteration of the loop.

Syntax

The basic syntax for creating a next statement in R is:

```
next
```

Flow Diagram



Example

```
v <- LETTERS[1:6]
for ( i in v){
if (i == "D"){
next
}
print(i)
}
```

When the above code is compiled and executed, it produces the following result:

```
[1] "A"
[1] "B"
[1] "C"
```

```
[1] "E"
[1] "F"
```

```
> v <- LETTERS[1:6]
> for ( i in v){
+     if (i == "D"){
+         next
+     }
+     print(i)
+ }
[1] "A"
[1] "B"
[1] "C"
[1] "E"
[1] "F"
>
```