

R Programming

Unit-I

- This will launch R interpreter and you will get a prompt `>` where you can start typing your program as follows –

```
>myString <- "Hello, World!"
```

```
> print ( myString)
```

- R does not support multi-line comments but you can perform a trick which is something as follows –

```
if(FALSE) { "This is a demo for multi-line  
  comments and it should be put inside either a  
  single OR double quote" }  
myString <- "Hello, World!"  
print ( myString)
```

- Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values.
- This means that, when you create a variable you reserve some space in memory.
- In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are –
 - Vectors
 - Lists
 - Matrices
 - Arrays
 - Factors
 - Data Frames

Data Type	Example	Verify
Logical	TRUE, FALSE	<div><div><code>v <- TRUE</code></div><div><code>print(class(v))</code></div></div> <div>Live Demo </div> <div>it produces the following result –</div> <div><code>[1] "logical"</code></div>
Numeric	12.3, 5, 999	<div><div><code>v <- 23.5</code></div><div><code>print(class(v))</code></div></div> <div>Live Demo </div> <div>it produces the following result –</div> <div><code>[1] "numeric"</code></div>
Integer	2L, 34L, 0L	<div><div><code>v <- 2L</code></div><div><code>print(class(v))</code></div></div> <div>Live Demo </div> <div>it produces the following result –</div> <div><code>[1] "integer"</code></div>

Complex	$3 + 2i$	<div> <div> <pre>v <- 2+5i</pre> <pre>print(class(v))</pre> </div> <div> Live Demo </div> </div> <p>it produces the following result –</p> <div> <pre>[1] "complex"</pre> </div>
Character	'a' , "good", "TRUE", '23.4'	<div> <div> <pre>v <- "TRUE"</pre> <pre>print(class(v))</pre> </div> <div> Live Demo </div> </div> <p>it produces the following result –</p> <div> <pre>[1] "character"</pre> </div>
Raw	"Hello" is stored as 48 65 6c 6c 6f	<div> <div> <pre>v <- charToRaw("Hello")</pre> <pre>print(class(v))</pre> </div> <div> Live Demo </div> </div> <p>it produces the following result –</p> <div> <pre>[1] "raw"</pre> </div>

- In R programming, the very basic data types are the R-objects called **vectors** which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types.
- For example, we can use many atomic vectors and create an array whose class will become array.

Vectors

- When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

Create a vector.

```
apple <- c('red','green',"yellow")  
print(apple)
```

Get the class of the vector.

```
print(class(apple))
```


Lists

- A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

Create a list.

```
list1 <- list(c(2,5,3),21.3,sin)
```

Print the list.

```
print(list1)
```

Matrices

- A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.
- # Create a matrix.

```
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3,  
            byrow = TRUE)  
print(M)
```

O/p:

	[, 1]	[, 2]	[, 3]
[1,]	"a"	"a"	"b"
[2,]	"c"	"b"	"a"

Arrays

- While matrices are confined to two dimensions, arrays can be of any number of dimensions.
- The array function takes a dim attribute which creates the required number of dimension.
- In the below example we create an array with two elements which are 3x3 matrices each.

Create an array.

```
a <- array(c('green','yellow'),dim = c(3,3,2))
```

```
print(a)
```


O/p :

, , 1

	[,1]	[,2]	[,3]
[1,]	"green"	"yellow"	"green"
[2,]	"yellow"	"green"	"yellow"
[3,]	"green"	"yellow"	"green"

, , 2

	[,1]	[,2]	[,3]
[1,]	"yellow"	"green"	"yellow"
[2,]	"green"	"yellow"	"green"
[3,]	"yellow"	"green"	"yellow"

Factors




- Factors are the r-objects which are created using a vector.
- It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector.
- They are useful in statistical modeling.

- `apple_colors <-
c('green','green','yellow','red','red','red','green'
)`

Operator

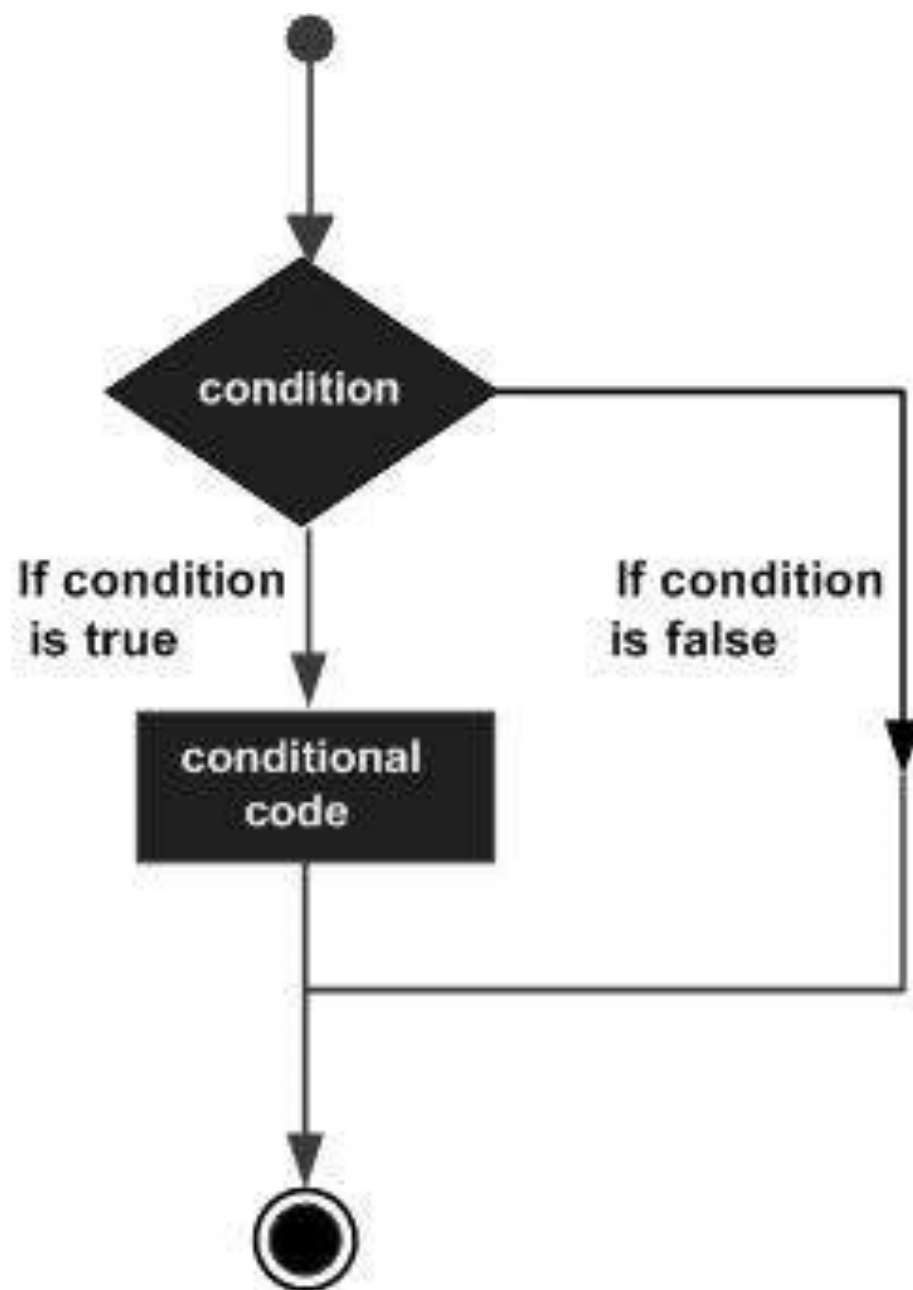
- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators. Types of operators are :
 - Arithmetic Operators
 - Relational Operators
 - Logical Operators
 - Assignment Operators
 - Miscellaneous Operators

Operator	Description	Example
+	Adds two vectors	<div> Live Demo <pre>v <- c(2,5.5,6) t <- c(8, 3, 4) print(v+t)</pre> </div> <p>it produces the following result –</p> <pre>[1] 10.0 8.5 10.0</pre>
-	Subtracts second vector from the first	<div> Live Demo <pre>v <- c(2,5.5,6) t <- c(8, 3, 4) print(v-t)</pre> </div> <p>it produces the following result –</p> <pre>[1] -6.0 2.5 2.0</pre>
*	Multiplies both vectors	<div> Live Demo <pre>v <- c(2,5.5,6) t <- c(8, 3, 4) print(v*t)</pre> </div> <p>it produces the following result –</p> <pre>[1] 16.0 16.5 24.0</pre>

/	Divide the first vector with the second	<div data-bbox="871 91 1734 239"> <pre>v <- c(2, 5.5, 6)</pre> <pre>t <- c(8, 3, 4)</pre> <pre>print(v/t)</pre> <div data-bbox="1483 91 1676 125">Live Demo </div> </div> <p>When we execute the above code, it produces the following result –</p> <div data-bbox="871 404 1734 472"> <pre>[1] 0.250000 1.833333 1.500000</pre> </div>
%%	Give the remainder of the first vector with the second	<div data-bbox="871 576 1734 725"> <pre>v <- c(2, 5.5, 6)</pre> <pre>t <- c(8, 3, 4)</pre> <pre>print(v%%t)</pre> <div data-bbox="1483 576 1676 611">Live Demo </div> </div> <p>it produces the following result –</p> <div data-bbox="871 843 1734 912"> <pre>[1] 2.0 2.5 2.0</pre> </div>
%/%	The result of division of first vector with second (quotient)	<div data-bbox="871 1015 1734 1163"> <pre>v <- c(2, 5.5, 6)</pre> <pre>t <- c(8, 3, 4)</pre> <pre>print(v%/%t)</pre> <div data-bbox="1483 1015 1676 1049">Live Demo </div> </div> <p>it produces the following result –</p> <div data-bbox="871 1282 1734 1350"> <pre>[1] 0 1 1</pre> </div>

Conditional Statements

- 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**.



- R provides the following types of decision making statements. Click the following links to check their detail.
- [if statement](#) : An **if** statement consists of a Boolean expression followed by one or more statements.
- [if...else statement](#) : An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.
- [switch statement](#) : A **switch** statement allows a variable to be tested for equality against a list of values.

- **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. }
```

- **Example**

```
x <- 30L
```

```
if(is.integer(x))
```

```
{
```

```
  print("X is an Integer")
```

```
}
```


- 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. }

Example

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

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

```
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")
}
```

Switch Statement

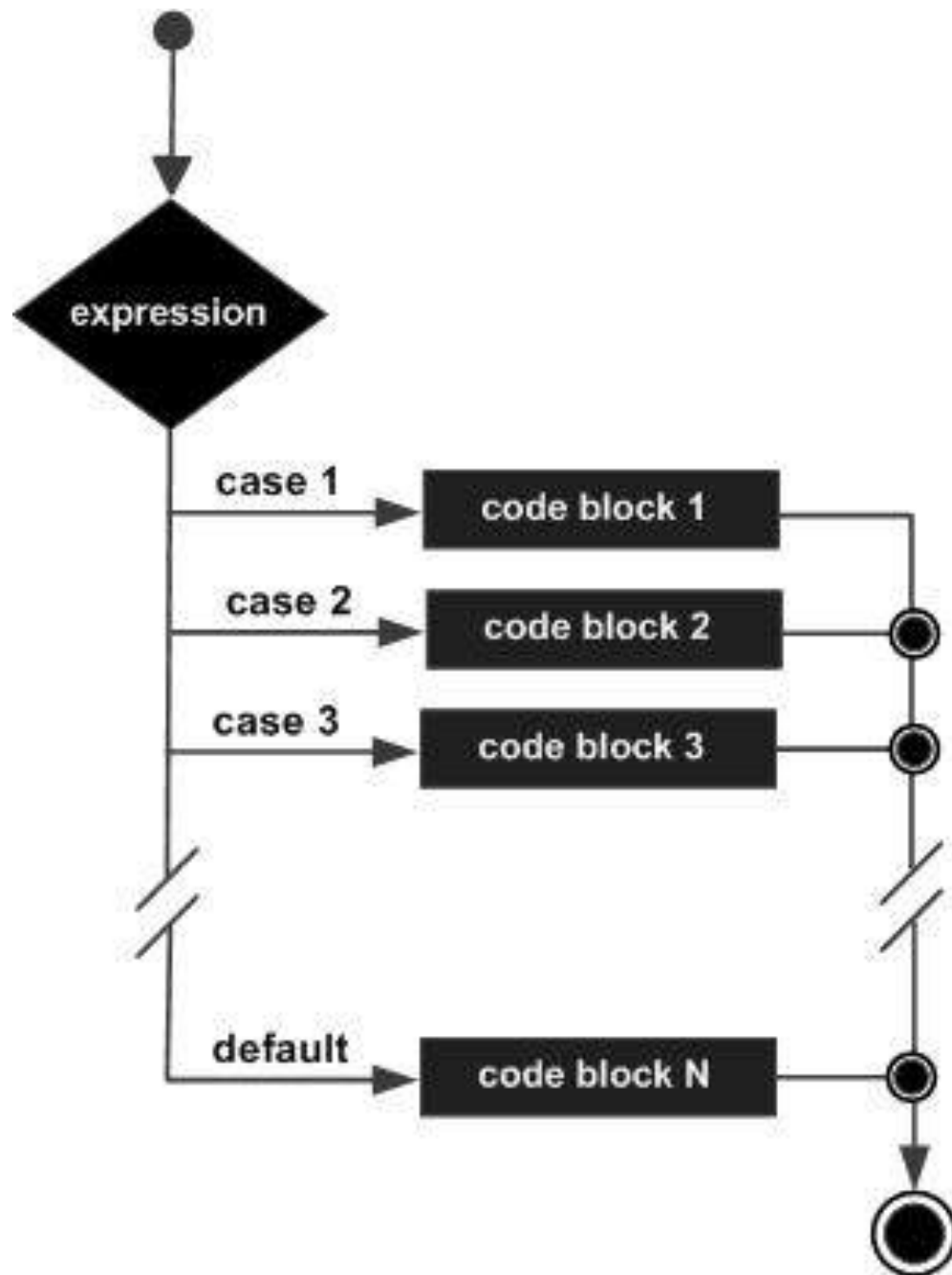
- A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax

```
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.)



Example

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

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.

The different types of loops control statements are :

- for loop
- nested loop
- while loop
- repeat and break statement
- return statement
- next statement

for loop

- It is a type of loop or sequence of statements executed repeatedly until exit condition is reached.

Syntax:

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

- **Example:**

```
x <- letters[4:10]
```

```
for(i in x){  
  print(i)  
}
```

Nested loops

- Nested loops are similar to simple loops. Nested means loops inside loop. Moreover, nested loops are used to manipulate the matrix.

```
# Defining matrix
```

```
m <- matrix(2:15, 2)
```

```
for (r in seq(nrow(m))) {  
  for (c in seq(ncol(m))) {  
    print(m[r, c])  
  }  
}
```

while loop

- while loop is another kind of loop iterated until a condition is satisfied. The testing expression is checked first before executing the body of loop.

- **Syntax:**

```
while(expression)
{
statement ....
}
```

- **Example:**

filter_none

brightness_4

`x = 1`

`# Print 1 to 5`

`while(x <= 5){`

`print(x)`

`x = x + 1`

`}`

repeat loop and break statement

- **repeat** is a loop which can be iterated many number of times but there is no exit condition to come out from the loop.
- So, break statement is used to exit from the loop. **break** statement can be used in any type of loop to exit from the loop.

Repeat

{

statements

if(expression)

{

Break

}

}

```
x = 1
```

```
# Print 1 to 5
```

```
repeat{
```

```
  print(x)
```

```
  x = x + 1
```

```
  if(x > 5){
```

```
    break
```

```
  }
```

```
}
```