Data types, Variables and Operators

Data types

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

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors.

Data Type	Example	Verify
Logical	TRUE, FALSE	v <- TRUE print(class(v)) it produces the following result − [1] "logical"
Numeric	12.3, 5, 999	v <- 23.5 print(class(v)) it produces the following result − [1] "numeric"
Integer	2L, 34L, 0L	v <- 2L print(class(v)) it produces the following result − [1] "integer"
Complex	3 + 2i	v <- 2+5i print(class(v)) it produces the following result − [1] "complex"
Character	'a' , '"good", "TRUE", '23.4'	v <- "TRUE" print(class(v)) it produces the following result − [1] "character"
Raw	"Hello" is stored as 48 65 6c 6c 6f	v <- charToRaw("Hello") print(class(v)) it produces the following result − [1] "raw"

Some useful functions

R provides many functions to examine features of vectors and other objects, for example

- class() - what kind of object is it (high-level)?
- typeof() - what is the object’s data type (low-level)?
- length() - how long is it? What about two dimensional objects?
- attributes() - does it have any metadata?

```
# Create a vector.
fruit <- c(‘apple',‘orange',“banana")
print(fruit)

# Get the class of the vector.
 print(class(apple))
```

Result –

```
[1] “apple”, “orange” ,“banana"
[1] "character“
```

We can also use logical indexing, negative indexing, and 0/1 to access the elements of a vector:

For example:

```
x <- c("Jan","Feb","March","Apr","May","June","July")
y <- x[c(TRUE,FALSE,TRUE,FALSE,FALSE,TRUE,TRUE)]
z <- x[c(-3,-7)]c <- x[c(0,0,0,1,0,0,1)] print(y) print(z) print(c)
```

[1] "Jan"   "March" "June" "July"(All TRUE values are printed)
[1] "Jan" "Feb" "Apr" "May" "June"(**All corresponding values for negative indexes are dropped**)
[1] "Jan" "Jan"(**All corresponding values are printed**)

Missing Data

R supports missing data in vectors. They are represented as NA (Not Available).

```
>x <- c(0.5, NA, 0.7)
>x <- c(TRUE, FALSE, NA)
>x <- c("a", NA, "c", "d", "e")
>x <- c(1+5i, 2-3i, NA)
```

The function is.na() indicates the elements of the vectors that represent missing data, and the function anyNA() returns TRUE if the vector contains any missing values.

Example:

```
x <- c("a", NA, "c", "d", NA)
y <- c("a", "b", "c", "d", "e")
 is.na(x)
anyNA(y)
```

Output:

```
[1] FALSE TRUE FALSE FALSE TRUE
[1] FALSE
```

Variable

- A variable in R can store an atomic vector, group of atomic vectors or a combination of many Robjects.
- A valid variable name consists of letters, numbers and the dot or underline characters.
- The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

Variable Assignment

- The variables can be assigned values using leftward, rightward and equal to operator.
- The values of the variables can be printed using **print()** or **cat()** function.
- The **cat()** function combines multiple items into a continuous print output.

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)
# Assignment using leftward operator.
var.2 <- c("learn","R")
# Assignment using rightward operator.
c(TRUE,1) -> var.3

print(var.1)
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

Output:-

```
[1] 0 1 2 3
var.1 is 0 1 2 3
var.2 is learn R
var.3 is 1 1
```

Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language

```
var_x <- "Hello"
cat("The class of var_x is ",class(var_x),"\n")
var_x <- 34.5
cat(" Now the class of var_x is ",class(var_x),"\n")
var_x <- 27L
cat(" Next the class of var_x becomes ",class(var_x),"\n")
```

Result:-

```
The class of var_x is character
Now the class of var_x is numeric
Next the class of var_x becomes integer
```

Finding Variables

ls() function is used to know all the variables currently available in workspace.
Also the ls() function can use patterns to match the variable names.

```
print(ls())               #Print all the variables available in workspace

print(ls(pattern = "var"))        # Print List the variables starting with the pattern "var".

print(ls(all.name = TRUE))        # Also show the variables starting with dot(.)  as
                                   they are hidden in the second command.
```

Deleting Variables

Variables can be deleted by using the **rm()** function.

```
rm(var_x)             # Delete variable var_x

rm(list = ls())       #Delete all variables
```

Types of Operators

We have the following types of operators in R programming −

- •Arithmetic Operators
- •Relational Operators
- •Logical Operators
- •Assignment Operators
- •Miscellaneous Operators

Arithmetic Operators

Operator	Description	Example

+	Adds two vectors	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v+t) it produces the following result −
		[1] 10.0 8.5 10.0

−	Subtracts second vector from the first	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v-t) it produces the following result −
		[1] -6.0 2.5 2.0

*	Multiplies both vectors	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v*t) it produces the following result –
		[1] 16.0 16.5 24.0

/	Divide the first vector with the second	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v/t) it produces the following result –
		[1] 0.250000 1.833333 1.500000

%%	Give the remainder of the first vector with the second	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v%%t) it produces the following result –
		[1] 2.0 2.5 2.0

%/%	The result of division of first vector with second (quotient)	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v%/%t) it produces the following result –
		[1] 0 1 1

^	The first vector raised to the exponent of second vector	v <- c(2,5.5,6)
		t <- c(8, 3, 4)
		print(v^t) it produces the following result –
		[1] 256.000 166.375 1296.000

Relational Operators

Operator	Description	Example

>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v>t) it produces the following result –
		[1] FALSE TRUE FALSE FALSE

- **Similarly <, <=, >= , ==, != operators works**

Logical Operators

Operator	Description	Example

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.	v <- c(3,1,TRUE,2+3i)
		t <- c(4,1,FALSE,2+3i)
		print(v&t) it produces the following result –
		[1] TRUE TRUE FALSE TRUE

- **Similarly |(OR), !(NOT) operators works**

The logical operator && and || **considers only the first element of the vectors and give a vector of single element as output.**

<table>
<tr><th>Operator</th><th>Description</th><th>Example</th></tr>
<tr><td>&&</td><td>Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.</td><td>v <- c(3,0,TRUE,2+2i)
t <- c(1,3,TRUE,2+3i)
print(v&&t)
it produces the following result –
[1] TRUE</td></tr>
<tr><td>||</td><td>Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.</td><td>v <- c(0,0,TRUE,2+2i)
t <- c(0,3,TRUE,2+3i)
print(v||t)
it produces the following result –
[1] FALSE</td></tr>
</table>

Assignment Operator

Operator	Description	Example

<–
or
=
or
<<–
	Called Left Assignment	v1 <- c(3,1,TRUE,2+3i)
		v2 <<- c(3,1,TRUE,2+3i)
		v3 = c(3,1,TRUE,2+3i)
		print(v1)
		print(v2)
		print(v3)
		it produces the following result –
		[1] 3+0i 1+0i 1+0i 2+3i
		[1] 3+0i 1+0i 1+0i 2+3i
		[1] 3+0i 1+0i 1+0i 2+3i

-> or ->>	Called Right Assignment	c(3,1,TRUE,2+3i) -> v1 c(3,1,TRUE,2+3i) ->> v2 print(v1) print(v2) it produces the following result – [1] 3+0i 1+0i 1+0i 2+3i [1] 3+0i 1+0i 1+0i 2+3i

Other Operators:- used for specific purpose and not general mathematical or logical computation

Operator	Description	Example

:	Colon operator. It creates the series of numbers in sequence for a vector.	v <- 2:8
		print(v)
		it produces the following result –
		[1] 2 3 4 5 6 7 8

%in%	This operator is used to identify if an element belongs to a vector.	v1 <- 8
		v2 <- 12
		t <- 1:10
		print(v1 %in% t)
		print(v2 %in% t)
		it produces the following result –
		[1] TRUE [1] FALSE

%*%	This operator is used to multiply a matrix with its transpose.	M = matrix(c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE) t = M %*% t(M) print(t) it produces the following result – [,1] [,2] [1,] 65 82 [2,] 82 117

Control Statements

- R provides the following types of statements.

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.

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.

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.

If Statement

Syntax:

```
if(Boolean_expression)
{     }
```

For example:

```
x <- “Intelligant”
if(is.character(x))
{   print("X is a Character")
}
```

Output:

```
[1] “X is a Character”
```

Else Statement

Syntax:

```
if(Boolean_expression)
{   }
else {   }
```

For example:

```
x <- c("Intelligant","R","Tutorial")
if("Intelligant" %in% x)
{   print("Intelligant") }
else
{   print("Not found") }
```

Output:

```
[1] “Intelligant”
```

Else If Statement

Syntax:

```
if(Boolean_expression1)
{   }
 else if(Boolean_expression2)
{    } else if(Boolean_expression3)
{    } else {    }
```

For example:

```
x <- c("Intelligant", "R", "Tutorial")
 if("Intelligant" %in% x)
{   print("Intelligant") }
else if ("Tutorial" %in% x)
{print("Tutorial") }
 else {   print("Not found")}
```

Output:

```
[1] “Intelligant”
```

Switch Statement

Syntax:

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

Example 1:

```
x <- switch(  3,   "Intelligant",   "R",   "Tutorial",   "Beginners" )
print(x)
```

Output:

```
[1] “Tutorial”
```

Example 2:

```
y <-  "12“
 x <- switch(    y,    "9"= "Good Morning",    "12"= "Good Afternoon",    "18"= "Good
Evening",    "21"= "Good Night" )
print(x)
```

Output:

```
[1] “Good Afternoon”
```

If an **expression evaluates to a character string**, then it is matched (exactly) to the names of the cases mentioned in the switch statement.

If there is more than one match, the first matching element is returned.

No default argument is available.

Repeat Loop

Syntax:

```
repeat
{
statements
if(exit_condition)
{ break }
}
```

Example:

```
v <- 9
Repeat
 {
print(v)
v=v-1
if(v < 1)
{ break }
 }
```

Output:

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

Note: If we don’t place a break condition in the repeat loop statement, the statements in the repeat block will get executed in an infinite loop.

While Loop

Syntax:

```
while (Boolean_expression)
 { statement }
```

Example:

```
v <-9
while(v>5)
{ print(v)
v = v-1
 }
```

Output:

```
[1] 9
[1] 8
[1] 7
[1] 6
```

For Loop

Syntax:

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

Example:

```
v <- c(1:5)
for (i in v)
{ print(i) }
```

Output:

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

Example 2:(Use of break)

```
v <- c(1:5)
 for (i in v)
{ if(i == 3)
{ break }
print(i) }
```

Output:

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

Break Statement

A break statement is used for two purposes

- To terminate a loop immediately and resume at the next statement following the loop.
- To terminate a case in a switch statement.

Example:

```
v <- c(0:6)
 for (i in v)
{ if(i == 3)
{ break }
print(i)
 }
```

Output:

```
[1] 0
[1] 1
[1] 2
```

Next Statement

A next statement is one of the control statements in R programming that is used to skip the current iteration of a loop without terminating the loop. Whenever a next statement is encountered, further evaluation of the code is skipped and the next iteration of the loop starts.

Example:

```
v <- c(0:6)
for (i in v)
{
 if(i == 3)
{ next }
print(i) }
```

Output:

```
[1] 0
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
```

Break and Next statements are known as **Loop Control statements**