R – Function

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Function Definition

An R function is created by using the keyword **function**. The basic syntax of an R function definition is as follows:

```
function_name <- function(arg_1, arg_2, ...) {
    Function body
}
```

Function Components

The different parts of a function are:

- **Function Name:** This is the actual name of the function. It is stored in R environment as an object with this name.
-
- **Arguments:** An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
-
- **Function Body:** The function body contains a collection of statements that defines what the function does.
-
- **Return Value:** The return value of a function is the last expression in the function body to be evaluated.

R has many **in-built** functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as **user defined** functions.

Built-in Function

Simple examples of in-built functions are seq(), mean(), max(), sum(x)and paste(...) etc. They are directly called by user written programs. You can refer most widely used R functions.

```
# Create a sequence of numbers from 32 to 44.
print(seq(32,44))

# Find mean of numbers from 25 to 82.
print(mean(25:82))

# Find sum of numbers frm 41 to 68.
print(sum(41:68))
```

When we execute the above code, it produces the following result:

```
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44
[1] 53.5
[1] 1526
```

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
  for(i in 1:a) {
            b <- i^2
            print(b)
            }
                     }
```

Calling a Function

```
 # Create a function to print squares of numbers in sequence.
```

```
new.function <- function(a) {
  for(i in 1:a) {
            b <- i^2
            print(b)
            }
                     }

# Call the function new.function supplying 6 as an argument.
new.function(6)
```

When we execute the above code, it produces the following result:

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

Calling a Function without an Argument

```
# Create a function without an argument.
new.function <- function() {
            for(i in 1:5) {
            print(i^2)
            }
                     }

# Call the function without supplying an argument.
new.function()
```

When we execute the above code, it produces the following result:

```
[1] 1
[1] 4
[1] 9
```

```
[1] 16
[1] 25
```

Calling a Function with Argument Values (by position and by name)

The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

```
# Create a function with arguments.
new.function <- function(a,b,c) {
            result <- a*b+c
            print(result)
                        }

# Call the function by position of arguments.
new.function(5,3,11)

# Call the function by names of the arguments.
new.function(a=11,b=5,c=3)
```

When we execute the above code, it produces the following result:

```
[1] 26
[1] 58
```

Calling a Function with Default Argument

We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result. But we can also call such functions by supplying new values of the argument and get non default result.

```
# Create a function with arguments.
new.function <- function(a = 3,b =6) {
            result <- a*b
            print(result)
                              }
```

```
# Call the function without giving any argument.
new.function()

# Call the function with giving new values of the argument.
new.function(9,5)
```

When we execute the above code, it produces the following result:

```
[1] 18
[1] 45
```

Lazy Evaluation of Function

Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.

```
# Create a function with arguments.
new.function <- function(a, b) {
        print(a^2)
        print(a)
        print(b)
                        }

# Evaluate the function without supplying one of the arguments.
new.function(6)
```

When we execute the above code, it produces the following result:

```
[1] 36
[1] 6
Error in print(b) : argument "b" is missing, with no default
```

R – Strings

Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.

Rules Applied in String Construction

- The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.
- Double quotes can be inserted into a string starting and ending with single quote.
- Single quote can be inserted into a string starting and ending with double quotes.
- Double quotes can not be inserted into a string starting and ending with double quotes.
- Single quote can not be inserted into a string starting and ending with single quote.

Examples of Valid Strings

Following examples clarify the rules about creating a string in R.

```
a <- 'Start and end with single quote'
print(a)

b <- "Start and end with double quotes"
print(b)

c <- "single quote ' in between double quotes"
print(c)

d <- 'Double quotes " in between single quote'
print(d)
```

When the above code is run we get the following output:

```
[1] "Start and end with single quote"
[1] "Start and end with double quotes"
```

```
[1] "single quote ' in between double quote"
[1] "Double quote \" in between single quote"
```

Examples of Invalid Strings

```
e <- 'Mixed quotes"
print(e)

f <- 'Single quote ' inside single quote'
print(f)

g <- "Double quotes " inside double quotes"
print(g)
```

When we run the script it fails giving below results.

```
...: unexpected INCOMPLETE_STRING

.... unexpected symbol
1: f <- 'Single quote ' inside

unexpected symbol
1: g <- "Double quotes " inside
```

String Manipulation

Concatenating Strings - paste() function

Many strings in R are combined using the **paste()** function. It can take any number of arguments to be combined together.

Syntax

The basic syntax for paste function is :

```
paste(..., sep = " ", collapse = NULL)
```

Following is the description of the parameters used:

- **...** represents any number of arguments to be combined.
-
- **sep** represents any separator between the arguments. It is optional.
-
- **collapse** is used to eliminate the space in between two strings. But not the space within two words of one string.

-

Example

```
a <- "Hello"
b <- 'How'
c <- "are you? "

print(paste(a,b,c))

print(paste(a,b,c, sep = "-"))

print(paste(a,b,c, sep = "", collapse = ""))
```

When we execute the above code, it produces the following result:

```
[1] "Hello How are you? "
[1] "Hello-How-are you? "
[1] "HelloHoware you? "
```

Formatting numbers & strings - format() function

Numbers and strings can be formatted to a specific style using **format()**function.

Syntax

The basic syntax for format function is :

```
format(x, digits, nsmall,scientific,width,justify = c("left", "right", "centre",
"none"))
```

Following is the description of the parameters used:

- **x** is the vector input.

-
- **digits** is the total number of digits displayed.
-
- **nsmall** is the minimum number of digits to the right of the decimal point.
-
- **scientific** is set to TRUE to display scientific notation.
-
- **width** indicates the minimum width to be displayed by padding blanks in the beginning.
-
- **justify** is the display of the string to left, right or center.

-

Example

```
# Total number of digits displayed. Last digit rounded off.
result <- format(23.123456789, digits = 9)
print(result)

# Display numbers in scientific notation.
result <- format(c(6, 13.14521), scientific = TRUE)
print(result)

# The minimum number of digits to the right of the decimal point.
result <- format(23.47, nsmall = 5)
print(result)

# Format treats everything as a string.
result <- format(6)
print(result)

# Numbers are padded with blank in the beginning for width.
result <- format(13.7, width = 6)
print(result)

# Left justify strings.
```

```
result <- format("Hello",width = 8, justify = "l")
print(result)

# Justfy string with center.
result <- format("Hello",width = 8, justify = "c")
print(result)
```

When we execute the above code, it produces the following result:

```
[1] "23.1234568"
[1] "6.000000e+00" "1.314521e+01"
[1] "23.47000"
[1] "6"
[1] "  13.7"
[1] "Hello   "
[1] " Hello  "
```

Counting number of characters in a string - ncahr() function

This function counts the number of characters including spaces in a string.

Syntax

The basic syntax for nchar() function is :

```
nchar(x)
```

Following is the description of the parameters used:

- **x** is the vector input.

Example

```
result <- nchar("Count the number of characters")
print(result)
```

When we execute the above code, it produces the following result:

```
[1] 30
```

Changing the case - toupper() & tolower() functions

These functions change the case of characters of a string.

Syntax

The basic syntax for toupper() & tolower() function is :

```
toupper(x)
tolower(x)
```

Following is the description of the parameters used:

- **x** is the vector input.

Example

```
# Changing to Upper case.
result <- toupper("Changing To Upper")
print(result)

# Changing to lower case.
result <- tolower("Changing To Lower")
print(result)
```

When we execute the above code, it produces the following result:

```
[1] "CHANGING TO UPPER"
[1] "changing to lower"
```

Extracting parts of a string - substring() function

This function extracts parts of a String.

Syntax

The basic syntax for substring() function is :

```
substring(x,first,last)
```

Following is the description of the parameters used:

- **x** is the character vector input.
- **first** is the position of the first character to be extracted.

- **last** is the position of the last character to be extracted.

Example

```
# Extract characters from 5th to 7th position.
result <- substring("Extract", 5, 7)
print(result)
```

When we execute the above code, it produces the following result:

```
[1] "act"
```