Advance Programming Lab(CS-3095)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering

Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

2 Credit

R Programming

Lab Contents

Sr #	Major and Detailed Coverage Area	Lab#
1	Function ❑ Function Definition ❑ Function Declaration ❑ Function Components ❑ Function Calling ❑ Named Arguments ❑ Default Value of Arguments ❑ Built-in Function, User-defined Function ❑ Recursion	3

Function Definition

A function is a set of statements organized together to perform a specific task. Functions are used to logically break our code into simpler parts which become easy to maintain and understand.

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 Declaration

The basic syntax of an R function definition is as follows:

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

- The reserved word **function** is used to declare a function.
- The statements within the curly braces form the body of the function. These braces are optional if the body contains only a single expression.
- Finally, this function object is given a name by assigning it to a variable, func_name.

Function Component

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.

Function Example

Example 1: function to print x raised to the power y

```
pow <- function(x, y)
{
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
```

Example 2: function for conversion from fahrenheit to kelvin

```
fahrenheit_to_kelvin <- function(temp_F)
{
  temp_K <- ((temp_F - 32) * (5 / 9)) + 273.15
  return(temp_K)
}
```

Note: In R, it is not necessary to include the return statement. R automatically returns whichever variable is on the last line of the body of the function. While in the learning phase, we will explicitly define the return statement.

Calling a Function

We can call the above function as follows.

Example 1: calling pow

```
pow(8, 2)
[1] "8 raised to the power 2 is 64"
```

```
pow(2, 8)
[1] "2 raised to the power 8 is 256"
```

Example 2: calling fahrenheit_to_kelvin

```
# freezing point of water
output = fahrenheit_to_kelvin(32)
print(output)
[1] 273.15

# boiling point of water
fahrenheit_to_kelvin(212)
[1] 373.15
```

```
# freezing point of water
print(fahrenheit_to_kelvin(32))
[1] 273.15
```

Lab Work

Write a function called fence that takes two vectors as arguments, called original and wrapper, and returns a new vector that has the wrapper vector at the beginning and end of the original.

Solution

```
best_practice <- c("Write", "programs", "for", "people", "not", "computers")
asterisk <- "***"  # R interprets a variable with a single value as a vector with one element.
print(best_practice, asterisk)

fence <- function(original, wrapper)
{
 answer <- c(wrapper, original, wrapper)
 return(answer)
}
```

Lab Work

Write a function called fence that takes one vector as argument, and that returns a vector made up of just the first and last elements of its input.

Solution

Named Arguments

In the function calls, the argument matching of formal argument to the actual arguments takes place in positional order. This means that, in the call pow(8,2), the formal arguments x and y are assigned 8 and 2 respectively.

We can also call the function using **named arguments**. When calling a function in this way, the order of the actual arguments doesn't matter. For example, all of the function calls given below are equivalent.

```
pow(8, 2)

[1] "8 raised to the power 2 is 64“

pow(x = 8, y = 2)

[1] "8 raised to the power 2 is 64“

pow(y = 2, x = 8)

[1] "8 raised to the power 2 is 64"
```

Named Arguments cont'd

Furthermore, we can use named and unnamed arguments in a single call.

In such case, all the named arguments are matched first and then the remaining unnamed arguments are matched in a positional order.

```
pow(x=8, 2)
[1] "8 raised to the power 2 is 64“
pow(2, x=8)
[1] "8 raised to the power 2 is 64"
```

In all the examples above, x gets the value 8 and y gets the value 2.

Default Value for Arguments

Default values to arguments can be assigned in a function in R. This is done by providing an appropriate value to the formal argument in the function declaration. Below is the function with a default value for y.

```
pow <- function(x, y = 2)
{
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
```

The use of default value to an argument makes it optional when calling the function.

```
pow(3)
[1] "3 raised to the power 2 is 9“
pow(3,1)
[1] "3 raised to the power 1 is 3"
```

Built-in Function

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.

Examples:

```
# 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))
```

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

Recursive Function

A function that calls itself is called a recursive function and this technique is known as recursion. This special programming technique can be used to solve problems by breaking them into smaller and simpler sub-problems.

```
# Recursive function to find factorial
recursive.factorial <- function(x)
{
    if (x == 0)
        return (1)
    else
        return (x * recursive.factorial(x-1))
}
# Call the function recursive.factorial supplying 5 as an argument.
recursive.factorial(5)
[1] 120
```

Thank You
End of Lab 3

Assignment

1. Write an R-script to evaluate average of 3 numbers using function
2. Write an R-script to find out the factorial of a number using function
3. Write an R-script to find out HCF and LCM of the given two numbers using function
4. Write an R-script to evaluate sum of the following series using recursive function 1+2+3+…….. +N
5. Write an R-script to display the reverse of the given no. using recursive function
6. Write an R-script to evaluate the simple interest of the given P, T, R using function, where function takes the default value for R
7. Write an R-script to convert decimal into binary using recursive function
8. Write an R-script to find the factorial of a number using recursive function
9. Write an R-script to find the Find Sum of Series $1^2+2^2+3^2+.....+n^2$ using recursive function
10. Write an R-script to develop a function that receives 5 numbers and display the sum, average and standard deviation of these numbers.

Assignment cont'd

11. Write an R-script to generate a set of numbers and run the numbers with the following built-in statistic functions.

Function	Description
mean(x)	Mean of the numbers in vector x
median(x)	Median of the numbers in vector x
var(x)	Variance of the numbers in vector x
sd(x)	Standard deviation of the numbers in vector x
scale(x)	Standard scores (z-scores) of the numbers in vector x
summary(x)	max(x) of the numbers in vector x min(x) of the numbers in vector x
rank(x)	Ranks of the numbers (in increasing order) in vector x.
quantile(x)	The 0^{th}, 25^{th}, 50^{th}, 75^{th}, and 100^{th} percentiles (i.e. the quartiles) of the numbers in vector x.