Session 2 : Control Structures, Using Functions in R and Getting Help in R

Jamuna S Murthy
Assistant Professor
Department of CSE

Academic Year - 2025-26

1 If-Else Statements in R

The `if-else` statement in R allows conditional execution of code blocks based on a given condition.

1.1 Syntax

```
if (condition) {

Code to execute if condition is TRUE

} else {

Code to execute if condition is FALSE

}
```

1.2 Example: Checking if a Number is Positive or Negative

```
n <- as.numeric(readline("Enter a number: "))
if (n > 0) {
print("The number is positive")
} else if (n < 0) {
print("The number is negative")
} else {
print("The number is zero")
}
```

1.3 Nested If-Else

You can nest `if-else` statements inside each other.

```
score <- as.numeric(readline("Enter your score: "))
if (score >= 90) {
print("Grade: A")
} else if (score >= 75) {
print("Grade: B")
} else if (score >= 50) {
print("Grade: C")
} else {
print("Grade: F")
}
```

2 While Loop in R

The `while` loop executes a block of code repeatedly as long as the condition remains TRUE.

2.1 Syntax

```
while (condition) {

Code to execute

}
```

2.2 Example: Printing Numbers from 1 to 5

```
i <- 1
while (i <= 5) {
print(i)
i <- i + 1  # Increment i
}
```

2.3 Using Break in While Loop

The `break` statement exits the loop immediately.

```
i <- 1
while (TRUE) {
print(i)
if (i == 5) {
break  # Exit loop when i equals 5
}
i <- i + 1
}
```

3 For Loop in R

The `for` loop iterates over a sequence (vector, list, etc.).

3.1 Syntax

```
for (variable in sequence) {

Code to execute

}
```

3.2 Example: Printing Elements of a Vector

```
numbers <- c(10, 20, 30, 40, 50)
for (num in numbers) {
print(num)
}
```

3.3 Using Break and Next in For Loop

`break` exits the loop.

`next` skips the current iteration and continues.

Example:

```
for (i in 1:10) {
if (i == 5) {
next  # Skip printing 5
}
print(i)
}
```

4 Practice Programs

1. Write an R program that takes a number as input and prints whether it is even or odd using `if-else`.
2. Create a while loop that prints numbers from 10 down to 1.
3. Write a for loop to calculate the sum of all even numbers from 1 to 50.
4. Use a for loop to print the first 10 terms of the Fibonacci sequence.
5. Write an R program that takes a number as input and checks if it is a prime number using a for loop.

5 Using Functions in R

Functions in R are used to encapsulate reusable pieces of code. They help in structuring the code, reducing redundancy, and improving readability.

5.1 Defining Functions

A function in R can be defined using the following syntax:

```
function_name <- function(arg1, arg2, ...) {

Function body

return(value)
}
```

Example:

```
add_numbers <- function(a, b) {
result <- a + b
return(result)
}

print(add_numbers(5, 3))  # Output: 8
```

5.2 Built-in Functions

R provides many built-in functions, such as:

- **mean()** - Calculates the mean of a numeric vector.
- **sum()** - Computes the sum of elements.
- **sqrt()** - Computes the square root.
- **length()** - Returns the number of elements in a vector.
- **abs()** - Returns the absolute value of a number.

Example:

```
numbers <- c(1, 2, 3, 4, 5)
print(mean(numbers))  # Output: 3
```

5.3 Function Arguments and Defaults

Functions in R can have default arguments. If a user does not pass a value, the default is used.

Example:

```
add_numbers <- function(a, b = 10) {
return(a + b)
}

print(add_numbers(5))  # Output: 15
print(add_numbers(5, 3))  # Output: 8
```

5.4 Anonymous Functions

Anonymous functions (also called lambda functions) can be defined and used without naming them.

Example:

```
(sapply(1:5, function(x) x^2))

Output: 1 4 9 16 25
```

5.5 Returning Multiple Values

A function in R can return multiple values as a list.

Example:

```
calculate <- function(a, b) {
sum <- a + b
product <- a * b
return(list(sum = sum, product = product))
}

result <- calculate(4, 5)
print(result$sum)  # Output: 9
print(result$product)  # Output: 20
```

6 Getting Help in R and Quitting RStudio

6.1 Getting Help in R

R provides built-in help mechanisms to understand functions and packages.

6.1.1 Using the Help System

```
help(mean)    # Help on the mean function
?mean         # Alternative way to get help
```

6.1.2 Searching for Help

If you are not sure about the function name, use:

```
apropos("mean")  # Lists all functions related to 'mean'
help.search("mean")  # Searches documentation for 'mean'
```

6.2 Quitting R and RStudio

To quit R, use the following commands in the console:

```
q()   # Prompts for confirmation before exiting
```

Alternatively, in RStudio, go to **File** → **Quit Session**.

7 Installing and Loading Packages in R

7.1 Installing Packages

R packages extend functionality. To install a package from CRAN, use:

```
install.packages("ggplot2")  # Installs ggplot2 package
```

7.2 Loading Packages

To use an installed package, load it into your session:

```
library(ggplot2)  # Loads the ggplot2 package
```

7.3 Checking Installed Packages

To see the list of installed packages:

```
installed.packages()
```

7.4 Updating Packages

To update all installed packages:

```
update.packages()
```

8 Practice Questions

1. Write an R function that takes a number as input and returns its square and cube.
2. Define an R function that takes a vector of numbers and returns the mean and median.
3. Write an R function to check if a number is prime.
4. Create an R function that calculates the factorial of a given number.
5. Write an R function that takes two numbers and returns a list containing their sum, difference, product, and quotient.