Kalinga University

Faculty of Computer Science & Information Technology

Course- BCAAIML
Subject- R Programming
Subject Code – BCAAIML401 **Sem- IV**

Unit 1

Introducing to R – R Data Structures – Help Functions in R – Vectors – Scalars – Declarations – Recycling – Common Vector Operations – Using all and any – Vectorized operations – NA and NULL values – Filtering – Victoriesed if-then else – Vector Element names.

Introduction to R and Data Structures

R is a programming language and software environment for statistical computing and graphics. It provides several built-in data structures, which are essential to handling and manipulating data effectively. The most common ones include:

- Vectors: A basic data structure that can hold elements of the same type (e.g., all numeric or all character).
- Matrices: Two-dimensional arrays, essentially vectors with a dimension attribute.
- Lists: Ordered collections of elements which can be of different types (heterogeneous).
- Data Frames: A special type of list where each element (column) is of equal length, commonly used for data manipulation.
- Factors: Used for categorical data.

Help Functions in R

In R, help functions are essential for exploring and understanding the functionality of any function or package. Some key help functions include:

help(function_name) or ?function_name: Displays documentation for a specific function.

- ?mean # Provides documentation for the mean function
- help.search("keyword"): Searches for a keyword across all documentation.
- help.search("vector") # Find all functions related to vectors
- str(object): Provides a compact, human-readable description of any object (e.g., a data frame, vector).
- str(mtcars) # Shows structure of the mtcars dataset

Vectors, Scalars, Declarations

In R, a vector is a sequence of elements of the same type (numeric, character, logical, etc.). A scalar is essentially a vector of length one.

- Vector Creation:
- Numeric Vector:

  v <- c(1, 2, 3, 4, 5) # Creates a numeric vector

- Character Vector:

  char_vec <- c("apple", "banana", "cherry")

- Logical Vector:

  logical_vec <- c(TRUE, FALSE, TRUE)

- Scalars (single-element vectors):

  scalar <- 42 # This is a scalar, technically a vector of length 1

Recycling and Common Vector Operations

In R, recycling is a process where R repeats the shorter vector to match the length of the longer one.

```
v <- c(1, 2, 3)
v2 <- c(10, 20)
v + v2  # Recycling occurs; it becomes: c(1+10, 2+20, 3+10)
```

Common vector operations include:

Arithmetic operations: +, -, *, /, etc.

Comparison operations: ==, !=, <, >, etc.

Logical operations: & (AND), | (OR), ! (NOT).

```
v <- c(1, 2, 3, 4)
v * 2  # Element-wise multiplication
```

Using all and any, Vectorized Operations

all(): Returns TRUE if all elements in a logical vector are TRUE.

```
all(c(TRUE, TRUE, TRUE))  # TRUE
all(c(TRUE, FALSE, TRUE)) # FALSE
```

any(): Returns TRUE if any element in a logical vector is TRUE.

```
any(c(FALSE, FALSE, TRUE))  # TRUE
any(c(FALSE, FALSE, FALSE)) # FALSE
```

R is designed to perform vectorized operations, meaning operations on entire vectors or arrays at once, rather than having to use loops. This makes R very efficient for numerical computations.

```
v <- c(1, 2, 3, 4)
v^2  # Returns: c(1, 4, 9, 16)
```

NA and NULL Values

NA: Represents missing values or undefined data.

```
v <- c(1, 2, NA, 4)
sum(v, na.rm = TRUE)  # Ignoring NA while summing
```

NULL: Represents the absence of a value or an undefined object. Often used to indicate an empty object.

```
v <- NULL
```

length(v) # Returns 0
You can check for these values using is.na() and is.null():

```
is.na(v)  # Check if element is NA
is.null(v)  # Check if object is NULL
```

Filtering Vectors

You can filter vectors using logical conditions or indices.

```
v <- c(1, 2, 3, 4, 5)
filtered_v <- v[v > 3]  # Filters values greater than 3
```

You can also filter using which() to get indices of elements that satisfy a condition.
which(v > 3) # Returns indices of elements greater than 3

Vectorized If-Then-Else
R allows you to use vectorized conditional operations with functions like ifelse().

```
v <- c(1, 2, 3, 4)
result <- ifelse(v %% 2 == 0, "Even", "Odd")
# Returns: c("Odd", "Even", "Odd", "Even")
```

Here, ifelse(condition, value_if_true, value_if_false) is used to return a vector of results based on a condition applied to each element.

Vector Element Names

You can assign names to vector elements, which can make your code more readable and allow you to access elements by name.

```
v <- c(10, 20, 30)
names(v) <- c("a", "b", "c")
v["a"]  # Accesses the element named "a"
```

This is useful for giving meaning to the elements, especially when working with datasets or results.

Example Summary: Using the Concepts Together

```
# Creating a vector and performing operations
vec <- c(10, 20, NA, 40)
# Filtering with NA removal
filtered_vec <- vec[!is.na(vec)]  # Removes NA

# Vectorized operation: Double the values
doubled_vec <- filtered_vec * 2

# Using ifelse for vectorized conditional logic
labels <- ifelse(doubled_vec > 50, "Large", "Small")

# Named vector elements
names(doubled_vec) <- c("First", "Second", "Fourth")

# Print results
print(doubled_vec)
print(labels)
```