Unit 2: R Data structures and Manipulation

Creating Variables, expressions, R data types and objects, Numeric, Character and Logical Data, Vectors, Scalars, Declarations, Common Vector operations, Conditional statements and loops, Arithmetic Operators, and Logical Operations. Reading datasets and exporting data from R, Manipulating and processing data in R.

1. Introduction to R Variables and Expressions

- In R, **variables** are used to store data.
- You can assign values using:
  - `<-` (preferred in R),
  - `=` (also valid),
  - `->` (less commonly used).
- **Examples**:
- `x <- 10`
- `y = 20`
- `30 -> z`

2. R Data Types and Objects

- **Basic data types** in R:
  - **Numeric**: Decimal or integer (e.g., `3.5`, `7`)
  - **Character**: Text strings (`"hello"`)
  - **Logical**: Boolean values (`TRUE`, `FALSE`)
- **Objects** in R include:
  - **Vector**, **Matrix**, **List**, **Data Frame**, **Array**, **Factor**
- **R Objects Overview**
- An object in R is a data structure that holds values. R is an object-oriented language, and **everything is treated as an object**.
- *Common R Objects:*

Object Type	Structure	Example
Vector	1D, homogeneous	`c(1, 2, 3)`
Matrix	2D, homogeneous	`matrix(1:9, nrow=3)`
Array	Multi-dimensional	`array(1:8, c(2, 2, 2))`
List	1D, heterogeneous	`list(1, "Aafrin", TRUE)`
Data Frame	2D, heterogeneous	`data.frame(Name, Age)`

Object Type	Structure	Example
Factor	Categorical variable	`factor(c("low", "high"))`

3. Scalars and Vectors

- **Scalar**: A vector of **length 1**
- **Vector**: A one-dimensional data structure of the same type
- Created using the `c()` function
- `numbers <- c(1, 2, 3, 4, 5)`
- Vectors can be numeric, character, or logical.

4. Common Vector Operations

- Arithmetic: `+`, `-`, `*`, `/`
- Logical: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Functions:
  - `length()`, `sum()`, `mean()`, `max()`, `min()`, `sort()`

Example:

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
z <- x + y  # z = c(5, 7, 9)
```

5. Conditional Statements and Loops

- **Conditional**: `if`, `else if`, `else`
- **Loops**:
  - `for`, `while`, `repeat`

✅ Conditional Statements and Loops in R

These are **control flow structures** that allow your program to make decisions and execute code multiple times based on conditions.

1. **Conditional Statements**

💡 if Statement

Executes a block of code if a condition is **TRUE**.

```
x <- 10
if (x > 5) {
  print("x is greater than 5")
```

```
}
```

if...else Statement

Executes one block if the condition is **TRUE**, another if it is **FALSE**.

```
x <- 3
if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is less than or equal to 5")
}
```

💡 if...else if...else Statement

Used to test multiple conditions.

```
x <- 0
if (x > 0) {
  print("Positive number")
} else if (x < 0) {
  print("Negative number")
} else {
  print("Zero")
}
```

◆ 2. Loops

Loops are used to execute a block of code repeatedly.

🔄 for Loop

Used to iterate over a sequence (like a vector).

```
for (i in 1:5) {
  print(i)
}
```

🔁 Example with Vector:

```
fruits <- c("apple", "banana", "cherry")
for (fruit in fruits) {
  print(fruit)
}
```

🔄 while Loop

Keeps executing as long as the condition is **TRUE**.

```
x <- 1
while (x <= 5) {
  print(x)
  x <- x + 1
}
```

⛔ repeat Loop

Runs infinitely unless stopped using `break`.

```
x <- 1
repeat {
  print(x)
  x <- x + 1
  if (x > 5) {
    break
  }
}
```

🔄 Loop Control Statements

Statement	Purpose
`break`	Exit the loop entirely
`next`	Skip current iteration and go to the next one

Example with `next`:

```
for (i in 1:5) {
  if (i == 3) {
    next
  }
  print(i)
}
```

Example with `break`:

```
for (i in 1:5) {
  if (i == 4) {
    break
  }
  print(i)
}
```

Summary Table:

Structure	Usage
`if`	Executes code if condition is true

Structure	Usage
`if...else`	Executes one of two blocks
`if...else if`	Multiple conditions
`for`	Loop through elements in a sequence
`while`	Loop while a condition is true
`repeat`	Infinite loop unless broken
`break`	Exit loop
`next`	Skip current iteration in a loop

6. Arithmetic and Logical Operators

- **Arithmetic**: `+`, `-`, `*`, `/`, `^` (power), `%%` (modulo), `%/%` (integer division)
- **Logical**:
  - Element-wise: `&`, `|`, `!`
  - Comparisons: `==`, `!=`, `>`, `<`, `>=`, `<=`

Example:

```
5 %% 2  # returns 1 (remainder)
TRUE & FALSE  # returns FALSE
```

7. Reading and Writing Data

- **Read**:
- `data <- read.csv("file.csv")`
- **Write**:
- `write.csv(data, "output.csv")`

8. Manipulating and Processing Data

- **Subsetting**:
- `data[1, ]  # First row`
- `data[, 2]  # Second column`
- `data[1:3, 2:4]  # Rows 1 to 3, Columns 2 to 4`
- **Filtering Rows**:
- `subset(data, Marks > 60)`

9. R Data Structures Overview

Structure	Dimensionality	Data Type	Example Use
Vector	1D	Homogeneous	Simple data list
Matrix	2D	Homogeneous	Numeric tables
Array	2D+	Homogeneous	Multi-dim data
List	1D	Heterogeneous	Mixed data
Data Frame	2D	Heterogeneous	Tabular data

---

10. Importing and Exporting Data

- **read.csv()**: Reads comma-separated values
- **read.table()**: Reads general table format
- **write.csv()**: Exports data frame to CSV

✅ Importing Data

You can **read data** into R using the following functions:

◆ read.csv()

- Used to read comma-separated value (CSV) files.
- Automatically sets header=TRUE by default.

Example:

```
data <- read.csv("students.csv")
```

◆ read.table()

- Used to read tabular data from text files.
- Requires you to define separators explicitly.

Example:

```
data <- read.table("students.txt", header=TRUE, sep="\t")
```

📤 Exporting Data

You can **write data** from R to external files using:

◆ write.csv()

- Writes data frames to a CSV file.

Example:

```
write.csv(data, "output.csv", row.names=FALSE)
```

◆ **write.table()**

- Writes a data frame or matrix to a text file with customizable separators.

Example:

```
write.table(data, "output.txt", sep="\t", row.names=FALSE)
```

◆ **write.xlsx() (from openxlsx or writexl packages)**

- Used to export data to Excel format.

```
library(openxlsx)

write.xlsx(data, "output.xlsx")
```