UNIT-I

Basics of R: Introduction, R-Environment Setup, Programming with R, Basic Data Types. **Vectors**: Creating and Naming Vectors, Vector Arithmetic, Vector Subsetting. **Matrices**: Creating and Naming Matrices, Matrix Subsetting. **Arrays**, **Class**.

---

1. What is R?

R is an **open-source programming language and software environment** used for:

- Statistical computing
- Data analysis
- Data visualization
- Scientific research

R was developed by **Ross Ihaka and Robert Gentleman** at the **University of Auckland, New Zealand**.

---

2. Why R is Used?

R is widely used because it provides:

- Powerful statistical functions
- Advanced graphical capabilities
- Thousands of ready-to-use packages
- Easy data handling and manipulation

It is especially popular among **data scientists, statisticians, researchers, and academicians**.

---

3. Features of R

- **Free and Open Source**
- **Platform Independent** (Windows, Linux, macOS)
- Supports **Object-Oriented Programming**
- Large package ecosystem (CRAN)
- Excellent **data visualization**
- Strong community support
- Interpreted language (no compilation needed)

---

4. Applications of R

R is used in many domains such as:

- **Data Science & Analytics**
- **Machine Learning**
- **Bioinformatics**
- **Finance & Banking**
- **Healthcare Analytics**
- **Academic Research**
- **Artificial Intelligence**

---

5. Advantages of R

- Easy to learn and use
- High-quality graphics and plots
- Large number of statistical techniques
- Easy integration with other languages (Python, C, Java)
- Frequent updates and improvements

---

6. Limitations of R

- Slower execution for very large datasets
- Memory-intensive (stores data in RAM)
- Not ideal for large-scale production systems
- Steep learning curve for beginners in programming

---

7. R as a Programming Language

- R is an **interpreted language**
- Commands are executed line by line
- Case-sensitive (x and X are different)
- Supports **procedural, functional, and OOP styles**

Example:

```
x <- 10
```

```
y <- 20
x + y
```

---

8. R Packages

- Packages are collections of functions
- Available through **CRAN**
- Examples: ggplot2, dplyr, tidyverse

Installing a package:

```
install.packages("ggplot2")
library(ggplot2)
```

---

9. R Environment Components

- **Console** – Executes commands
- **Script Editor** – Write programs
- **Workspace** – Stores objects
- **Packages** – Additional functionalities

---

10. Comparison: R vs Other Languages

Feature	R	Python
Primary use	Statistics	General purpose
Visualization	Excellent	Good
Learning curve	Moderate	Easy
Packages	CRAN	PyPI

Vectors

1. What is a Vector in R?

A **vector** is the **most basic data structure in R**.
It is a **one-dimensional collection of elements** of the **same data type**.

✓ All elements in a vector must be **homogeneous** (same type).

Feature	R	Python

2. Creating Vectors in R

2.1 Using c() Function (Combine)

```
v1 <- c(10, 20, 30, 40)
v2 <- c("R", "Python", "Java")
v3 <- c(TRUE, FALSE, TRUE)
```

Check type:

```
class(v1)
```

2.2 Using : Operator (Sequence)

```
v <- 1:10
```

2.3 Using seq() Function

```
v <- seq(from = 1, to = 10, by = 2)
```

2.4 Using rep() Function

```
v <- rep(5, times = 4)
v2 <- rep(c(1, 2), times = 3)
```

3. Naming Vectors

3.1 Naming During Creation

```
marks <- c(Maths = 85, Physics = 78, Chemistry = 90)
```

3.2 Naming After Creation

```
marks <- c(85, 78, 90)
names(marks) <- c("Maths", "Physics", "Chemistry")
```

Access named elements:

```
marks["Maths"]
```

Feature	R	Python

3.3 Removing Names

```
names(marks) <- NULL
```

4. Vector Arithmetic

R supports **element-wise operations** on vectors.

4.1 Addition, Subtraction, Multiplication, Division

```
a <- c(10, 20, 30)
b <- c(1, 2, 3)

a + b
a - b
a * b
a / b
```

4.2 Scalar Operations

```
a <- c(10, 20, 30)
a + 5
a * 2
```

4.3 Vector Recycling Rule

If vectors are of different lengths, R **recycles** elements.

```
x <- c(1, 2, 3, 4)
y <- c(10, 20)

x + y
```

4.4 Logical Operations on Vectors

```
x <- c(5, 10, 15)
x > 8
```

Feature	R	Python

5. Vector Subsetting (Accessing Elements)

Subsetting means **selecting specific elements** from a vector.

5.1 By Index (Position)

```
v <- c(10, 20, 30, 40, 50)

v[1]      # First element
v[3]      # Third element
v[c(2,4)] # Multiple elements
```

5.2 Negative Indexing (Excluding Elements)

```
v[-1]      # Exclude first element
v[-c(2,3)] # Exclude 2nd and 3rd elements
```

5.3 Logical Subsetting

```
v <- c(10, 20, 30, 40)

v[v > 25]
```

5.4 Subsetting Using Names

```
marks <- c(Maths = 85, Physics = 78, Chemistry = 90)

marks["Physics"]
```

5.5 Using which() Function

```
v <- c(5, 12, 18, 7)

v[which(v > 10)]
```

Feature	R	Python

6. Useful Vector Functions

Function	Purpose
length()	Number of elements
sum()	Sum of elements
mean()	Average
min()	Minimum
max()	Maximum
sort()	Sorting
unique()	Unique values

Example:

```
v <- c(10, 20, 20, 30)
unique(v)
```

7. Type Coercion in Vectors

```
v <- c(10, "R", TRUE)
class(v)
```

✓ All elements are converted to **character**

Matrices

1. What is a Matrix in R?

A **matrix** is a **two-dimensional data structure** in R that stores elements in **rows and columns**.

✓ All elements in a matrix must be of the **same data type** (homogeneous).

---

2. Creating Matrices in R

2.1 Using matrix() Function

Syntax:

matrix(data, nrow, ncol, byrow = FALSE)

Example:

m1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

m1

(Default filling is **column-wise**)

2.2 Creating Matrix by Row

m2 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, byrow = TRUE)

2.3 Creating Matrix Using Vectors

v1 <- c(1, 2, 3)

v2 <- c(4, 5, 6)

m <- cbind(v1, v2) # Column bind

m

m <- rbind(v1, v2) # Row bind

m

2.4 Creating Special Matrices

matrix(0, nrow = 3, ncol = 3) # Zero matrix

matrix(1, nrow = 2, ncol = 2) # Ones matrix

---

3. Naming Rows and Columns of a Matrix

3.1 Naming During Creation

m <- matrix(

 c(10, 20, 30, 40),

 nrow = 2,

 dimnames = list(

```
  c("Row1", "Row2"),
  c("Col1", "Col2")
 )
)
```

3.2 Naming After Creation

```
rownames(m) <- c("R1", "R2")
colnames(m) <- c("C1", "C2")
```

3.3 Viewing Names

```
rownames(m)
colnames(m)
```

3.4 Removing Names

```
rownames(m) <- NULL
colnames(m) <- NULL
```

---

4. Accessing Matrix Elements (Matrix Subsetting)

Matrix subsetting uses:

```
matrix[row, column]
```

4.1 Access Single Element

```
m <- matrix(1:9, nrow = 3)

m[2, 3]   # 2nd row, 3rd column
```

4.2 Access Entire Row or Column

```
m[1, ]    # First row
m[, 2]    # Second column
```

4.3 Access Multiple Rows and Columns

```
m[c(1,3), c(2,3)]
```

4.4 Excluding Rows or Columns (Negative Indexing)

```
m[-1, ]   # Remove first row

m[, -2]   # Remove second column
```

4.5 Subsetting Using Names

```
rownames(m) <- c("A", "B", "C")

colnames(m) <- c("X", "Y", "Z")


m["B", "Y"]
```

4.6 Logical Subsetting

```
m[m > 5]
```

---

5. Matrix Operations (Basic)

5.1 Element-wise Operations

```
m1 <- matrix(1:4, nrow = 2)

m2 <- matrix(5:8, nrow = 2)


m1 + m2

m1 * m2
```

5.2 Matrix Multiplication

```
m1 %*% m2
```

5.3 Transpose of Matrix

t(m1)

---

6. Useful Matrix Functions

Function	Purpose
dim()	Dimensions
nrow()	Number of rows
ncol()	Number of columns
rowSums()	Sum of rows
colSums()	Sum of columns
apply()	Apply function

Example:

rowSums(m)

---

7. Type Coercion in Matrices

m <- matrix(c(1, 2, "R", 4), nrow = 2)

class(m)

✓ All values become **character**

Arrays

1. What is an Array in R?

An **array** is a **multidimensional data structure** in R that can store data in **more than two dimensions**.

- A **vector** → 1D
- A **matrix** → 2D
- An **array** → 2D or more (3D, 4D, etc.)

✓ All elements in an array must be of the **same data type** (homogeneous).

---

2. Difference Between Matrix and Array

Feature	Matrix	Array
Dimensions	2D	2D or more
Rows & Columns	Yes	Yes
Layers	No	Yes
Data type	Homogeneous	Homogeneous

---

3. Creating Arrays in R

3.1 Using array() Function

Syntax:

array(data, dim, dimnames = NULL)

3.2 Creating a 2D Array

a <- array(1:6, dim = c(2, 3))

a

✓ Filled **column-wise** by default.

3.3 Creating a 3D Array

a3 <- array(1:12, dim = c(2, 3, 2))

a3

- 2 rows
- 3 columns
- 2 matrices (layers)

3.4 Creating Array Using Vectors

v <- c(10, 20, 30, 40)

arr <- array(v, dim = c(2, 2, 1))

arr

---

4. Naming Dimensions of an Array

4.1 Naming While Creation

```
arr <- array(
  1:8,
  dim = c(2, 2, 2),
  dimnames = list(
    c("R1", "R2"),
    c("C1", "C2"),
    c("Layer1", "Layer2")
  )
)
arr
```

4.2 Naming After Creation

```
dimnames(arr) <- list(
  c("Row1", "Row2"),
  c("Col1", "Col2"),
  c("M1", "M2")
)
```

---

5. Accessing Elements in an Array (Subsetting)

General Format:

array[row, column, layer]

5.1 Access a Single Element

```
arr[1, 2, 1]
```

5.2 Access Entire Row, Column, or Layer

```
arr[1, , ]    # First row
arr[ , 2, ]   # Second column
arr[ , , 1]   # First layer
```

5.3 Access Multiple Elements

arr[c(1,2), c(1,2), 1]

5.4 Subsetting Using Names

arr["Row1", "Col2", "M1"]

5.5 Logical Subsetting

arr[arr > 5]

---

6. Array Operations

6.1 Element-wise Arithmetic

a1 <- array(1:8, dim = c(2,2,2))

a2 <- array(9:16, dim = c(2,2,2))

a1 + a2

a1 * a2

6.2 Apply Functions on Arrays

apply(arr, c(1), sum) # Row-wise

apply(arr, c(2), mean) # Column-wise

apply(arr, c(3), sum) # Layer-wise

---

7. Useful Array Functions

Function	Purpose
dim()	Dimensions
length()	Total elements
dimnames()	Dimension names
apply()	Apply function
str()	Structure

Example:

dim(arr)

8. Type Coercion in Arrays

arr <- array(c(1, 2, "R", 4), dim = c(2,2))

class(arr)

✓ All values become **character**

Class

1. What is a Class in R?

In R, a **class** defines the **type and structure of an object**.
It tells R **how data should be treated** and **which methods (functions) apply to it**.

🔷 Every object in R belongs to at least **one class**.

Example:

x <- 10

class(x)

Output:

[1] "numeric"

---

2. Importance of Class in R

- Determines how functions behave on an object
- Enables **Object-Oriented Programming (OOP)**
- Helps in **data organization**
- Improves code reusability and readability

---

3. Built-in Classes in R

Data	Class
10	numeric
10L	integer
"R"	character

Data	Class
TRUE	logical
2+3i	complex
c(1,2,3)	numeric
list()	list
matrix()	matrix
data.frame()	data.frame
factor()	factor

---

4. Checking and Assigning Class

4.1 Check Class

```
x <- c(1, 2, 3)
class(x)
```

4.2 Assign a Class

```
class(x) <- "MyClass"
class(x)
```

4.3 Multiple Classes

```
class(x) <- c("MyClass", "numeric")
```

---

5. Object-Oriented Programming Systems in R

R supports **four OOP systems**:

OOP System	Description
S3	Simple, informal, most commonly used
S4	Formal, strict definitions
R6	Reference-based, modern OOP
RC	Older reference classes

---

6. S3 Class (Most Important for Exams)

6.1 Creating an S3 Class

Step 1: Create Object

```
student <- list(
  name = "Ravi",
  age = 20,
  branch = "CSE"
)
```

Step 2: Assign Class

```
class(student) <- "Student"
```

6.2 Constructor Function

```
Student <- function(name, age, branch) {
  obj <- list(
    name = name,
    age = age,
    branch = branch
  )
  class(obj) <- "Student"
  return(obj)
}

s1 <- Student("Anita", 21, "IT")
```

---

7. Methods in S3 Class

A **method** is a function that works for a specific class.

Example: print() Method

```
print.Student <- function(x) {
  cat("Student Details\n")
  cat("Name:", x$name, "\n")
  cat("Age:", x$age, "\n")
```

```
  cat("Branch:", x$branch, "\n")
}

print(s1)
```

---

8. Generic Functions and Method Dispatch

- print() is a **generic function**
- R automatically calls print.ClassName()

```
print(s1)   # Calls print.Student()
```

---

9. S4 Class (Brief Overview)

S4 classes are **formal and strict**.

Example:

```
setClass(
  "Employee",
  slots = list(
    id = "numeric",
    name = "character",
    salary = "numeric"
  )
)

emp <- new("Employee", id = 101, name = "Ravi", salary = 50000)
```

---

10. R6 Class (Brief Overview)

R6 uses **reference semantics**.

```
library(R6)

Person <- R6Class("Person",
  public = list(
    name = NULL,
```

```
  initialize = function(name) {
    self$name <- name
  },
  greet = function() {
    cat("Hello,", self$name)
  }
 )
)

p <- Person$new("Ravi")
p$greet()
```

---

11. Class vs Data Type

Feature	Data Type	Class
Meaning	Kind of data	Structure + behavior
Example	numeric	data.frame
Purpose	Storage	OOP

Difference Between S3 and R6 Classes

Feature	S3 Class	R6 Class
OOP type	Informal	Formal
Definition	No strict definition	Strict class definition
Object behavior	Copy-on-modify	Modified in place
Method calling	print(obj)	obj$method()
Package required	No	Yes (R6)
Ease of use	Very easy	Moderate