Lab Sheet 01

Part 01: Fundamentals

1. Printing and Comments

Let's start by printing some values and adding comments to explain the code.

```
# Printing a number (Here the comment is shown with #)

print(200)


# Printing a string

print("Dog")
```

---

2. Mathematical Operations

Perform basic arithmetic operations such as addition, multiplication, division, exponentiation, remainder, and integer division.

```
# Arithmetic operations

print(1 + 2)  # Addition

print(2 * 3)  # Multiplication

print(4 / 2)  # Division

print(4 ^ 2)  # Exponentiation using ^

print(4 ** 2) # Exponentiation using **

print(5 %% 2) # Remainder

print(5 %/% 2) # Integer division
```

---

3. Relational Operations

Compare values using relational operators like greater than, less than, equal to, and not equal to.

```
# Relational operations
print(20 > 10)   # Greater than
print(30 < 20)   # Less than
print(50 >= 30)  # Greater than or equal to
print(10 == 10)  # Equal
print(10 != 10)  # Not equal
```

---

4. Logical Operations

Combine conditions using logical operators such as AND, OR, and NOT.

```
# Logical operations
print((20 > 10) && (30 <= 50))  # AND
print((20 < 10) || (20 == 20))  # OR
print(!(10 == 10))              # NOT
```

---

5. Variables

Define and manipulate variables, and check which variables exist in the environment.

```
# Assign values
x = 10
y = 20


# Addition
z = x + y
```

```
print(z)


# List variables
print(ls())


# Remove variable x
rm(x)
print(ls())


# Remove all variables
rm(list = ls())
print(ls())
```

---

6. Mathematical Functions

Explore built-in mathematical functions like summation, absolute value, square root, logarithms, exponentials, and rounding.

```
# Mathematical functions
print(sum(10, 20))
print(abs(-20))
print(sqrt(25))
print(log(10))  # Natural log
print(exp(5))   # Exponential
print(round(3.245))
print(round(10.35, 1))
```

---

7. Primitive Data Types

Explore different data types in R, such as numeric, integer, character, logical, and complex. Verify their types using the class() function.

```
# Data types

num = 1.23

int = 12L

char = "Cat"

log = TRUE

comp = 23 + 3i


# Check types

print(class(num))

print(class(int))

print(class(char))

print(class(log))

print(class(comp))
```

---

8. Casting Data Types

Convert between data types, such as numeric to character, character to numeric, and logical to numeric.

```
# Casting data types

x = 12

y = as.character(x)

print(y)

print(class(y))
```

```
x = "12"
y = as.numeric(x)
print(y)
print(class(y))

x = TRUE
y = as.numeric(x)
print(y)
print(class(y))
```

---

9. User Input

Accept user input and process it, such as converting age input from character to numeric.

```
# User input
name = readline(prompt = "Enter your name: ")
age = readline(prompt = "Enter your age: ")

print(name)
print(age)

# Convert age to numeric
age = as.numeric(age)
print(age)
```

Part 02: Compound Data Structures

---

1. Vectors

Creating Vectors

1. Create numeric, character, and mixed vectors using c(). Observe coercion when mixing types.
2. Generate sequences using 1:10, seq(), and rep().

```
# Examples of vector creation
numeric_vector = c(12, 22, 23, 34, 35)
character_vector = c("Dog", "Cat", "Rat")
mixed_vector = c(12, "Man", 23)
sequence_vector = seq(10, 50, by = 5)
repeated_vector = rep("Dog", 5)
```

Indexing and Slicing

Access specific elements, slices, and apply Boolean masking.

```
# Indexing and slicing
vector = c(23, 33, 34, 32, 45, 50, 65)
element = vector[1]
slice = vector[1:3]
boolean_masked = vector[vector > 40]
```

Element-Wise Operations

Perform arithmetic and comparisons on vectors.

```
# Element-wise operations
vector1 = c(2, 3, 4)
vector2 = c(3, 4, 5)
```

```
sum_vector = vector1 + vector2
comparison = vector1 > 2
```

Summary Functions

Explore functions like sum(), mean(), sd(), and range().

```
# Summary functions
vector = c(2, 3, 4, 5)
print(sum(vector))
print(mean(vector))
print(sd(vector))
print(range(vector))
```

Other Vector Functions

Use append(), sort(), unique(), and more for practical operations.

```
# Additional vector functions
vector = c(20, 10, 20, 30)
appended_vector = append(vector, 40)
sorted_vector = sort(vector)
unique_values = unique(vector)
sampled_values = sample(vector, 2)
```

---

2. Matrices

Creating Matrices

1. Create matrices using matrix(). Explore row-wise and column-wise filling.

```
# Matrix creation
matrix1 = matrix(1:6, nrow = 3, ncol = 2)
```

```
matrix2 = matrix(1:6, nrow = 3, ncol = 2, byrow = TRUE)
```

Matrix Properties

Determine dimensions, structure, and summaries.

```
# Matrix properties
print(dim(matrix1))
print(summary(matrix1))
```

Merging Matrices

Combine matrices using cbind() and rbind().

```
# Merging matrices
matrix1 = matrix(1:4, nrow = 2)
matrix2 = matrix(5:8, nrow = 2)
horizontal_merge = cbind(matrix1, matrix2)
vertical_merge = rbind(matrix1, matrix2)
```

Matrix Indexing and Operations

Access rows, columns, and perform matrix arithmetic.

```
# Matrix indexing
matrix1 = matrix(1:9, nrow = 3)
print(matrix1[1, ])  # First row
print(matrix1[, 2])  # Second column
print(matrix1[1:2, 2:3])


# Matrix operations
matrix1 = matrix(c(1, 2, 3, 4), nrow = 2)
matrix2 = matrix(c(5, 6, 7, 8), nrow = 2)
```

```
matrix_addition = matrix1 + matrix2
matrix_subtraction = matrix1 - matrix2
matrix_multiplication = matrix1 * matrix2  # Element-wise
matrix_dot_product = matrix1 %*% matrix2  # Matrix multiplication
matrix_transpose = t(matrix1)
print(matrix_addition)
print(matrix_transpose)
```

Advanced Matrix Operations

Calculate determinants, inverse, and diagonal elements.

```
# Determinants and inverse
matrix3 = matrix(c(4, 7, 2, 6), nrow = 2)
det_value = det(matrix3)
inverse_matrix = solve(matrix3)
print(det_value)
print(inverse_matrix)


# Diagonal elements
diag_elements = diag(matrix3)
print(diag_elements)
```

---

3. Factors

Unordered Factors

Convert categorical data into factors.

```
# Unordered factors
```

```
gender = factor(c("Male", "Female", "Male"))
print(gender)
```

Ordered Factors

Specify an order for factor levels.

```
# Ordered factors
levels = factor(c("Low", "Medium", "High"), levels = c("Low", "Medium", "High"), ordered = TRUE)
print(levels)
```

---

4. Lists

Creating Lists

Combine multiple data types into a list.

```
# List creation
my_list = list(
  vector = c(1, 2, 3),
  matrix = matrix(1:4, nrow = 2),
  char = "Hello"
)
print(my_list)
```

---

5. Data Frames

Manual Creation

Create data frames from vectors.

```
# Data frame creation
```

```
name = c("Kane", "Jane", "David")
age = c(23, 33, 34)
marks = c(89, 78, 88)
df = data.frame(name, age, marks)
print(df)
```

Accessing Columns

Access columns using [] and $.

```
# Accessing columns
print(df["name"])
print(df$name)
```

Indexing and Slicing

Access specific rows and columns.

```
# Indexing and slicing
print(df[1, 1])
print(df[1:2, ])
```

Modifying Data Frames

Change elements, add, and remove columns.

```
# Modifying data frames
df$grade = ifelse(df$marks > 80, "A", "B")
print(df)
```

Boolean Masking

Filter rows based on conditions.

```
# Boolean masking
filtered_df = df[df$marks > 80, ]
```

```
print(filtered_df)
```

Data Frame Functions

Explore head(), tail(), dim(), nrow(), ncol(), colnames(), rownames(), rowSums(), colSums(), rowMeans(), colMeans(), summary(), and str().

```
# Data frame functions
print(head(df))
print(tail(df))
print(dim(df))
print(nrow(df))
print(ncol(df))
print(colnames(df))
print(rownames(df))
print(rowSums(df[, c("age", "marks")]))
print(colSums(df[, c("age", "marks")]))
print(rowMeans(df[, c("age", "marks")]))
print(colMeans(df[, c("age", "marks")]))
print(summary(df))
print(str(df))
```

Changing Columns to Factors

Convert categorical columns to factors in a data frame.

```
# Changing to factors
df$name = factor(df$name)
print(str(df))
```

Working with CSV Files

Import and explore data from CSV files.

```
# Reading CSV files
data = read.csv("data.csv")
print(head(data))
```

Additional Data Frame Operations

Perform more column operations such as creating new columns based on conditions, renaming columns, and applying functions across rows or columns.

```
# Adding new columns
data$new_column = data$marks * 2
print(data)

# Renaming columns
colnames(data)[colnames(data) == "marks"] = "Scores"
print(colnames(data))

# Applying a function
data$scaled_marks = scale(data$Scores)
print(data)
```

Using the Apply Family of Functions

Apply functions to rows or columns using apply(), lapply(), and sapply().

```
# Apply functions
data$average_score = apply(data[, c("age", "Scores")], 1, mean)
print(data)
```

```
column_sums = sapply(data[, c("age", "Scores")], sum)
print(column_sums)

list_structure = lapply(data[, c("age", "Scores")], summary)
print(list_structure)
```