Basic R Syntax and Variables

1. Theoretical Aspects

Introduction to R Syntax

- R is a case-sensitive language designed for statistical computing and graphics.
- The structure of R code involves functions, arguments, and operators.
- Comments start with `#`, and anything following `#` is ignored by the interpreter.

Example:

```
# This is a comment
print("Hello, R!")  # This will display "Hello, R!"
```

Variable Assignment

- Variables are used to store data and are created using the assignment operators `<-` or `=`.
- Variable names must start with a letter and can contain letters, numbers, dots, or underscores.

Example:

```
x <- 10      # Assigns 10 to the variable x
y = 20       # Alternative assignment
sum <- x + y # Assigns the sum of x and y to the variable sum
```

Operators

Arithmetic Operators:

- `+`, `-`, `*`, `/`, `%%` (modulus), `^` (exponentiation).

Comparison Operators:

- `==`, `!=`, `<`, `>`, `<=`, `>=`.

Logical Operators:

- `&` (and), `|` (or), `!` (not).

Example:

```
a <- 15
b <- 4
result <- a + b     # Arithmetic: Adds a and b
is_equal <- a == b  # Comparison: Checks if a equals b
logical_test <- a > 10 & b < 5  # Logical: Both conditions must be true
```

2. Code Examples

Assigning Values

```
age <- 25
name <- "Alice"
is_student <- TRUE
print(age)        # Outputs: 25
print(name)       # Outputs: "Alice"
print(is_student) # Outputs: TRUE
```

Using Operators

```
num1 <- 8
num2 <- 3

# Arithmetic Operations
addition <- num1 + num2  # 11
division <- num1 / num2  # 2.6667
remainder <- num1 %% num2  # 2

# Comparison
comparison <- num1 > num2  # TRUE

# Logical
logical_check <- num1 > 5 & num2 < 5  # TRUE
```

Variable Reassignment

```
score <- 50
score <- score + 20  # Updates score to 70
```

Combining Strings

```
first_name <- "John"
last_name <- "Doe"
full_name <- paste(first_name, last_name)  # Outputs: "John Doe"
```

3. Best Practices

1. Use meaningful variable names (`sales_data`, not `x`).

2. Prefer `<-` over `=` for assignments.
3. Avoid using reserved words (e.g., `if`, `TRUE`) as variable names.
4. Use consistent naming conventions (e.g., `snake_case` or `camelCase`).
5. Comment your code to improve readability.
6. Initialize variables before use to avoid unintended behavior.
7. Use spaces around operators for better readability (`a <- b + c`, not `a<-b+c`).
8. Avoid hard-coding values; use variables instead.
9. Validate input values before using them in operations.
10. Use `rm()` to remove unused variables and free memory.

4. Exercises

1. Assign your age to a variable and print it.
2. Create a variable `x` and assign it the sum of 15 and 20. Print the value of `x`.
3. Write a script to check if a number stored in a variable `num` is greater than 10.
4. Concatenate two strings, `first` and `last`, into a full name.
5. Create a variable `score` with a value of 80. Increment it by 10 and print the result.
6. Check if the number 8 is divisible by 3 and print the remainder.
7. Write a script to test if a variable `y` is equal to 100.
8. Assign a logical value `TRUE` to a variable and negate it using the `!` operator.
9. Compare two variables, `a` and `b`, to check if one is greater than the other.
10. Create variables `p` and `q` with values 50 and 20. Calculate their product.
11. Assign a floating-point number to a variable and print its integer part using `floor()`.
12. Create a variable `z` that stores the square of a number `n`.
13. Write a script that checks if a variable `temp` is between 20 and 30.
14. Create a variable `speed` and set its value to 60. Write a script to double its value.
15. Combine three strings into one sentence using the `paste()` function.