Getting Started with R

Part 1

Table of contents

1 Programming in R 2

2 Why R? 3

3 Variables 3

4 Naming Variables 3

5 Assignment 3

6 Displaying Variable Value 4

7 Data Types 4

8 Accessing Data Elements 5

9 Checking Object Type 7

10 Changing Object Type 8

11 Special Values 8

12 Factors 9

13 Size of Objects 9

14 Mathematical Operations 10

15 Order of Operations 10

16 Logical Operations 11

17 Vector and Matrix Operations **11**

18 Useful Functions **12**

19 Sorting and Ranking **13**

1 Programming in R

- **Programming:** Writing instructions for a computer to perform specific tasks.
- **R Language:** A language and environment designed for statistical computing and graphics.

2 Why R?

- **Comprehensive Statistical Analysis**: `R` provides a wide array of statistical tests, models, and analyses.
- **Rich Visualization Libraries**: Libraries like `ggplot2` allow for sophisticated data visualizations.
- **Open Source**: Free to use, and benefits from a large community that contributes packages and updates.
- **Extensible**: Over **15,000** packages in the `CRAN` repository for various applications.
- **Data Handling Capabilities**: `R` can process both structured and unstructured data.
- **Platform Independent**: Runs on various **operating systems**.

Notable Companies Using R: Google, Facebook, Airbnb, Uber, and many more use `R` for data analysis.

3 Variables

- **Definition:** A storage area in programming to hold and manipulate data.
- **Importance:** Allows for data storage, retrieval, and manipulation.
- **Analogy:** Think of variables as labeled storage boxes.

4 Naming Variables

- Begin with a letter
- Avoid spaces (use underscores)
- Case-sensitive.

```
age <- 25
student_name <- "John"
pi_value = 3.14
```

5 Assignment

Storing a value inside a variable.

```r
x <- 5          # Preferred in R
total = 100     # Also works
7 -> z          # Rare
```

6 Displaying Variable Value

- Type the variable name, or

```r
x
```

```
[1] 5
```

- Use the **`print()`** function

```r
print(total)
```

```
[1] 100
```

7 Data Types

Classifications of data based on its nature.

```r
number = 5                          # Numeric (Scalar)
number
```

```
[1] 5
```

```r
messsage = "Hello"                  # Character (String)
messsage
```

```
[1] "Hello"
```

```r
flag = TRUE                         # Logical
flag
```

```
[1] TRUE
```

```r
grades_vector = c(90, 85, 88)     # Vector
grades_vector
```

```
[1] 90 85 88
```

```r
matrix_data = matrix(1:6, nrow=2) # Matrix
matrix_data
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```r
students_df = data.frame(Name=c("Anna", "Bob"),
                         Age=c(23, 25))  # Dataframe
students_df
```

```
  Name Age
1 Anna  23
2  Bob  25
```

```r
info = list(Name="John", Scores=c(90, 85, 88))  # List
info
```

```
$Name
[1] "John"

$Scores
[1] 90 85 88
```

8 Accessing Data Elements

Methods to extract specific data or subsets from data structures.

- **Using Square Brackets:**

1. For **vectors**: Extract specific elements

```r
third_grade = grades_vector[3]
third_grade
```

```
[1] 88
```

2. For **matrices**: Extract rows, columns, or individual elements.

```r
first_row = matrix_data[1,]
first_row
```

```
[1] 1 3 5
```

```r
second_column = matrix_data[,2]
second_column
```

```
[1] 3 4
```

```r
element_1_2 = matrix_data[1,2]
element_1_2
```

```
[1] 3
```

3. For **data frames**: Extract rows, columns, or specific data.

```r
Anna_data = students_df[1,]
Anna_data
```

```
  Name Age
1 Anna  23
```

```r
Age_column = students_df[, "Age"]
Age_column
```

```
[1] 23 25
```

4. Exclude specific elements using negative indices:

```
all_but_third = grades_vector[-3]
all_but_third
```

```
[1] 90 85
```

5. Access rows using boolean logic:

```
students_above_23 = students_df[students_df$Age > 23,]
students_above_23
```

```
  Name Age
2  Bob  25
```

- **Using $ for Data Frames:** To access specific columns by name.

```
ages = students_df$Age
ages
```

```
[1] 23 25
```

9 Checking Object Type

- **Purpose**: To identify the data type or structure of an object.
- **Function**: `class()`

```
class(number)
```

```
[1] "numeric"
```

```
class(grades_vector)
```

```
[1] "numeric"
```

```
class(students_df)
```

```
[1] "data.frame"
```

10 Changing Object Type

- **Purpose**: To convert data from one type to another.
- **Functions**: `as.numeric()`, `as.character()`, `as.logical()`, etc.

```
number
```

```
[1] 5
```

```
class(number)
```

```
[1] "numeric"
```

```
converted_number = as.character(number)
converted_number
```

```
[1] "5"
```

```
class(converted_number)
```

```
[1] "character"
```

- **Q**: What will happen when running the following line?

```
converted_number + 5
```

- **A**: Error in `converted_number + 5`: ! non-numeric argument to binary operator

11 Special Values

- `NA`: Missing data
- `Inf`: Infinity e.g., `10/0`
- `NaN`: Result of invalid operations e.g., `0/0`
- `NULL`: Absence of a value

12 Factors

Data type for categorical data

```
gender = factor(c("male", "female", "male"))
gender
```

```
[1] male   female male
Levels: female male
```

```
levels(gender)
```

```
[1] "female" "male"
```

- **Question**: what will be the output of the following? `as.numeric(gender)`
- **Answer**: `2, 1, 2`

13 Size of Objects

- **Purpose**: Determine dimensions or length.

```
length(grades_vector)
```

```
[1] 3
```

```
nrow(students_df)
```

```
[1] 2
```

```
ncol(students_df)
```

```
[1] 2
```

```
dim(students_df)
```

```
[1] 2 2
```

14 Mathematical Operations

- `+`: Addition
- `-`: Subtraction
- `*`: Multiplication
- `/`: Division
- `^`: Exponentiation (raising to a power)
- `%%`: Modulus (remainder after division)

```
result_add = 3 + 5       # Addition
result_sub = 8 - 3       # Subtraction
result_mul = 4 * 7       # Multiplication
result_div = 8 / 2       # Division
result_exp = 2^3         # Exponentiation
result_mod = 8 %% 3      # Modulus
```

15 Order of Operations

PEMDA Rule:

- `P`: Parentheses - Always start with operations inside parentheses or brackets.
- `E`: Exponents - Next, handle powers and square root operations.
- `MD`: Multiplication and Division - Process them as they appear from left to right.
- `AS`: Addition and Subtraction - Handle them last, moving from left to right.

- **Q**: `3 + 5 * 2`
- **A**: 13
- **Q**: `(3 + 5) * 2`
- **A**: 16
- **Q**: `2 ^ 2 * 3`
- **A**: 12
- **Tip**: Always use parentheses for clarity, even if not strictly needed.

16 Logical Operations

Operations that return `TRUE` or `FALSE` based on certain conditions:

- `==`: Equal to
- `!=`: Not equal to
- `<`: Less than
- `>`: Greater than
- `<=`: Less than or equal to
- `>=`: Greater than or equal to
- `&`: Logical `AND`
- `|`: Logical `OR`
- `!`: Logical `NOT`

```
(3 < 4) & (7 > 6)
(3 > 4) | (7 > 6)
!(5 == 5)
```

17 Vector and Matrix Operations

```
grades_vector
```

```
[1] 90 85 88
```

```
grades_vector + c(5, 5, 5)
```

```
[1] 95 90 93
```

```
grades_vector * 1.1
```

```
[1] 99.0 93.5 96.8
```

```
matrix_data
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```r
matrix_data * 2
```

```
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    4    8   12
```

```r
matrix_data %*% t(matrix_data)
```

```
     [,1] [,2]
[1,]   35   44
[2,]   44   56
```

18 Useful Functions

```r
paste("Hello", "World!", sep=" ")
```

```
[1] "Hello World!"
```

```r
c("apple"=5, "banana"=10)
```

```
 apple banana
     5     10
```

```r
colnames(students_df)
```

```
[1] "Name" "Age"
```

```r
min(grades_vector)
```

```
[1] 85
```

```r
highest_grade = max(grades_vector)
highest_grade
```

```
[1] 90
```

```r
which(grades_vector == highest_grade)
```

```
[1] 1
```

19 Sorting and Ranking

1. `sort()`: Organize elements in ascending or descending order.

```r
numbers = c(5, 2, 9, 3)
sort(numbers)
```

```
[1] 2 3 5 9
```

2. `order()`: Returns the indices that would arrange the data into ascending or descending order.

```r
names = c("Vicky", "Cristina", "Barcelona")
order(names)
```

```
[1] 3 2 1
```

3. `rank()`: Provides the rank of each element when the data is sorted. In case of ties, it assigns the average rank.

```r
scores = c(85, 95, 85, 90)
rank(scores)
```

```
[1] 1.5 4.0 1.5 3.0
```

Briefly,

- `sort()` directly arranges the data.
- `order()` provides indices for the arranged data.
- `rank()` gives the position of each data point in the sorted order.