Introduction to R

Tan Sein Jone

University of British Columbia

August 7, 2024

Table of contents

Table of Contents

Functional Programming

- Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- It is a declarative type of programming style.
- It focuses on what to solve rather than how to solve.
- It uses expressions instead of statements.
- It is based on mathematical functions.

Immutability

- In functional programming, data is immutable.
- This means that once a value is assigned to a variable, it should not be changed.
- This makes it easier to reason about the code and prevents bugs caused by side effects.

Pure Functions

- A pure function is a function where the output value is determined by its input values, without observable side effects.
- This is how functions in math work: Math.cos(x) will, for the same value of x, always return the same result.
- Pure functions are easier to reason about and test.

```
pure_function <- function(x, y){
    return(x + y)
}

impure_function <- function(x, y){
    print(x)
    return(x + y)
}
```

Breakdown

- The pure_function function takes two arguments, x and y, and returns their sum.
- The impure_function function takes two arguments, x and y, prints the value of x, and returns their sum.
- The pure_function function is a pure function because it only depends on its input values.
- The impure_function function is an impure function because it has a side effect (printing the value of x).

First Class Functions

- In functional programming, functions are first-class citizens.
- This means that functions can be assigned to variables, passed as arguments, and returned from other functions.
- This allows for the creation of higher-order functions.

```
add <- function(x, y){
    return(x + y)
}

subtract <- function(x, y){
    return(x - y)
}

operate <- function(func, x, y){
    return(func(x, y))
}

operate(add, 5, 3)
operate(subtract, 5, 3)
```

Higher Order Functions

- Higher-order functions are functions that can either take other functions as arguments or return them as results.
- This is possible because functions are first-class citizens.
- Higher-order functions allow us to abstract over actions, not just values.

```
add <- function(x, y){
    return(x + y)
}

subtract <- function(x, y){
    return(x - y)
}

create_operator <- function(op){
    if(op == "add"){
        return(add)
    } else if(op == "subtract"){
        return(subtract)
    }
}

operator <- create_operator("add")
operator(5, 3)
```

Recursion

- Recursion is a technique in which a function calls itself to solve a problem.
- Recursion is a common feature of functional programming.
- Recursion is used to solve problems that can be broken down into smaller subproblems.

```
factorial <- function(n){
    if(n == 0){
        return(1)
    } else {
        return(n * factorial(n - 1))
    }
}

factorial(5)
```

Breakdown

- The factorial function takes an integer n as an argument.
- If n is 0, the function returns 1.
- Otherwise, the function returns n times the factorial of n - 1.
- This continues until n is 0, at which point the function returns 1.
- The function is called with the argument 5, which returns 120.

Tail Recursion

- Tail recursion is a special form of recursion where the recursive call is the last thing the function does.
- Tail recursion is more efficient than regular recursion because it can be optimized by the compiler.
- Tail recursion is a common feature of functional programming.

Map, Filter, and Reduce

- Map, filter, and reduce are three common higher-order functions in functional programming.
- Map applies a function to each element of a list.
- Filter selects elements from a list based on a condition.
- Reduce combines all elements of a list into a single value.

Map

- The map function applies a function to each element of a list and returns a new list with the results.
- The map function is a higher-order function because it takes a function as an argument.
- The map function is a common feature of functional programming.

```
add_one <- function(x){
    return(x + 1)
}

numbers <- c(1, 2, 3, 4, 5)
mapped_numbers <- lapply(numbers, add_one)
```

Breakdown

- The add_one function takes an integer x as an argument and returns x + 1.
- The numbers vector contains the integers 1, 2, 3, 4, and 5.
- The lapply function applies the add_one function to each element of the numbers vector and returns a new list with the results.
- The mapped_numbers list contains the integers 2, 3, 4, 5, and 6.

Filter

- The filter function selects elements from a list based on a condition and returns a new list with the selected elements.
- The filter function is a higher-order function because it takes a function as an argument.
- The filter function is a common feature of functional programming.

```
is_even <- function(x){
    return(x %% 2 == 0)
}

numbers <- c(1, 2, 3, 4, 5)
filtered_numbers <- numbers[numbers %% 2 == 0]
```

Breakdown

- The is_even function takes an integer x as an argument and returns TRUE if x is even and FALSE otherwise.
- The numbers vector contains the integers 1, 2, 3, 4, and 5.
- The filtered_numbers vector contains the even integers from the numbers vector.

Reduce

- The reduce function combines all elements of a list into a single value using a binary operation.
- The reduce function is a higher-order function because it takes a function as an argument.
- The reduce function is a common feature of functional programming.

```
add <- function(x, y){
    return(x + y)
}

numbers <- c(1, 2, 3, 4, 5)
reduced_number <- Reduce(add, numbers)
```

Breakdown

- The add function takes two integers, x and y, as arguments and returns their sum.
- The numbers vector contains the integers 1, 2, 3, 4, and 5.
- The reduced_number variable contains the sum of all the integers in the numbers vector.

Functional Programming in R

- R is a functional programming language.
- R has functions and packages that make functional programming easier.
- R has higher-order functions like lapply, sapply, and Reduce that allow you to apply functions to lists and vectors.

Table of Contents

Data Handling

- Data handling is a crucial part of data analysis.
- R has a wide range of functions and packages that make data handling easier.

Data Structures

- R has several data structures that are used to store data.
- The most common data structures are vectors, matrices, data frames, and lists.

Lists

- A list is a one-dimensional array that can hold numeric, character, or logical data.
- Lists are created using the list() function.
- Lists can hold different types of data in each element.

```
list_1 <- list(1, "a", TRUE)
list_2 <- list(
    name = "Alice",
    age = 25,
    married = TRUE
)
```

Dictionaries

- A dictionary is a one-dimensional array that can hold key-value pairs.
- Dictionaries are created using the list() function.
- Dictionaries are similar to lists, but they have named elements.

Vectors

- A vector is a one-dimensional array that can hold numeric, character, or logical data.
- Vectors are created using the c() function.
- Vectors can be of two types: atomic vectors and lists.

```
numeric_vector <- c(1, 2, 3, 4, 5)
character_vector <- c("a", "b", "c", "d", "e")
logical_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
```

Atomic Vectors

- An atomic vector is a vector that can hold only one type of data.
- Atomic vectors can be of four types: numeric, character, logical, and complex.
- Atomic vectors are created using the c() function.

```
numeric_vector <- c(1, 2, 3, 4, 5)
character_vector <- c("a", "b", "c", "d", "e")
logical_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
complex_vector <- c(1 + 2i, 3 + 4i, 5 + 6i)
```

When To Use Vectors

- Use vectors when you have a one-dimensional array of data.
- Use atomic vectors when you have a one-dimensional array of data of the same type.
- Use lists when you have a one-dimensional array of data of different types.
- Use dictionaries when you have a one-dimensional array of key-value pairs.

Matrices

- A matrix is a two-dimensional array that can hold numeric, character, or logical data.
- Matrices are created using the matrix() function.
- Matrices are created by combining vectors.

```
matrix_1 <- matrix(1:9, nrow = 3, ncol = 3)
matrix_2 <- matrix(letters[1:9], nrow = 3, ncol = 3)
matrix_3 <- matrix(
    c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE),
    nrow = 2, ncol = 3
)
```

When To Use Matrices

- Use matrices when you have a two-dimensional array of data.
- Use matrices when you have numeric, character, or logical data.
- Use matrices when you want to perform matrix operations like addition, subtraction, multiplication, and division.
- Use matrices when you want to represent data in a tabular format.

Data Matrix Operations

- R has functions and packages that allow you to perform matrix operations.
- The most common matrix operations are addition, subtraction, multiplication, and division.
- R has functions that allow you to perform these operations on matrices.
- R has functions that allow you to transpose, invert, and concatenate matrices.
- R has functions that allow you to extract rows and columns from matrices.

```
matrix_1 <- matrix(1:9, nrow = 3, ncol = 3)
matrix_2 <- matrix(9:1, nrow = 3, ncol = 3)

# Addition
added_matrix <- matrix_1 + matrix_2

# Subtraction
subtracted_matrix <- matrix_1 - matrix_2

# Multiplication
multiplied_matrix <- matrix_1 %*% matrix_2

# Division
divided_matrix <- matrix_1 / matrix_2
```

Data Frames

- A data frame is a two-dimensional array that can hold numeric, character, or logical data.
- Data frames are created using the data.frame() function.
- Data frames are similar to matrices, but they can hold different types of data in each column.

```
data_frame <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)
```

When To Use Data Frames

- Use data frames when you have a two-dimensional array of data.
- Use data frames when you have different types of data in each column.
- Use data frames when you want to perform data manipulation tasks like filtering, sorting, and aggregating data.
- Use data frames when you want to represent data in a tabular format.

Data Frame Operations

- R has functions and packages that allow you to perform data frame operations.
- The most common data frame operations are filtering, sorting, and aggregating data.
- R has functions that allow you to perform these operations on data frames.
- R has functions that allow you to merge and join data frames.

```
data <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

# Filter data
filtered_data <- data[data$age > 30, ]

# Sort data
sorted_data <- data[order(data$age), ]

# Aggregate data
aggregated_data <- aggregate(
    data$age,
    by = list(data$married),
    FUN = mean
)
```

File IO

- R has functions that allow you to read and write data from and to files.
- The most common file formats are CSV, Excel, and text files.
- R has functions that allow you to read and write data in these formats.

```
data <- read.csv("data_frame.csv")
write.csv(data, "data_frame.csv")
```

Word of Caution

- If you specify the same file name for the read and write functions, the file will be overwritten.
- Make sure to back up your data before using the write function.
- Make sure to check the file permissions before using the write function.

Data Manipulation

- Data manipulation is the process of transforming data to make it more useful for analysis.
- R has functions and packages that make data manipulation easier.
- The most common data manipulation tasks are filtering, sorting, and aggregating data.

```
data <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

# Filter data
filtered_data <- data[data$age > 30, ]

# Sort data
sorted_data <- data[order(data$age), ]

# Aggregate data
aggregated_data <- aggregate(
    data$age,
    by = list(data$married),
    FUN = mean
)
```

Merge and Join

- Merge and join are two common data manipulation tasks.
- Merge is used to combine two data frames based on a common column.
- Join is used to combine two data frames based on a common column.

```
data_1 <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

data_2 <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    salary = c(50000, 60000, 70000)
)

merged_data <- merge(data_1, data_2, by = "name")
joined_data <- merge(
    data_1, data_2, by = "name",
    all = TRUE
)
```

Breakdown

- The data_1 data frame contains the name, age, and married columns.
- The data_2 data frame contains the name and salary columns.
- The merged_data data frame contains the name, age, married, and salary columns.
- The joined_data data frame contains the name, age, married, and salary columns.

Tidyverse

- Tidyverse is a collection of R packages that make data manipulation easier.
- Tidyverse packages are designed to work together and follow a consistent design philosophy.
- Tidyverse packages are widely used in the R community.

Breakdown

- The dplyr package provides functions for data manipulation tasks like filtering, sorting, and aggregating data.
- The tidyr package provides functions for data manipulation tasks like reshaping and tidying data.
- The ggplot2 package provides functions for data visualization tasks like creating plots and charts.
- The readr package provides functions for reading and writing data from and to files.

dplyr

- The dplyr package provides functions for data manipulation tasks like filtering, sorting, and aggregating data.
- Some common dplyr functions are filter(), arrange(), and summarise().
- filter() is used to filter rows based on a condition.
- arrange() is used to sort rows based on a column.
- summarise() is used to aggregate data based on a column.

```
library(dplyr)

data <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

filtered_data <- data %>% filter(age > 30)
sorted_data <- data %>% arrange(age)
aggregated_data <- data %>% summarise(mean_age = mean(ag
```

Breakdown

- The data data frame contains the name, age, and married columns.
- The filtered_data data frame contains the rows where the age column is greater than 30.
- The sorted_data data frame contains the rows sorted by the age column.
- The aggregated_data data frame contains the mean age of the data frame.

tidyr

- The tidyr package provides functions for data manipulation tasks like reshaping and tidying data.
- Some common tidyr functions are gather() and spread().
- gather() is used to reshape data from wide to long format.
- spread() is used to reshape data from long to wide format.

```
library(tidyr)

data <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

gathered_data <- data %>% gather(key = "variable", value
spreaded_data <- gathered_data %>% spread(key = "variabl
```

Breakdown

- The data data frame contains the name, age, and married columns.
- The gathered_data data frame contains the data in long format.
- The spreaded_data data frame contains the data in wide format.

ggplot2

- The ggplot2 package provides functions for data visualization tasks like creating plots and charts.
- The ggplot2 package is based on the grammar of graphics.
- The ggplot2 package allows you to create complex plots with a few lines of code.

```
library(ggplot2)

data <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

ggplot(data, aes(x = name, y = age)) +
    geom_bar(stat = "identity")
```

Breakdown

- The data data frame contains the name, age, and married columns.
- The ggplot function creates a plot with the data data frame.
- The aes function specifies the x and y variables for the plot.
- The geom_bar function creates a bar plot with the data.

readr

- The readr package provides functions for reading and writing data from and to files.
- The readr package is faster and more user-friendly than the base R functions.
- The readr package is widely used in the R community.

```
library(readr)

data <- read_csv("data_frame.csv")
write_csv(data, "data_frame.csv")
```

Breakdown

- The read_csv function reads data from a CSV file.
- The write_csv function writes data to a CSV file.
- The data data frame contains the data read from the CSV file.

Summary

- Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions.
- Functional programming focuses on immutability, pure functions, and first-class functions.
- R is a functional programming language that has functions and packages that make functional programming easier.
- R has data structures like vectors, matrices, data frames, and lists that are used to store data.
- R has functions and packages that allow you to perform data manipulation tasks like filtering, sorting, and aggregating data.