Internship

Mridul Jain

2023-05-08

Contents

Day 1: Recap of R

Lesson - Day 1
Exercises - Day 1

Exercise 1: OOPs

S3:

```
# Creating Student
MakeStudentS3 <- function(n, no, mM, mP, mC){
  students3 <- list(name = n, rollNo = no, marksInMath = mM, marksInPhysics = mP, marksInChemistry = mC)
  class(students3) <- "StudentS3"
  return(students3)
}

# Print Function
print.StudentS3 <- function(students3){
  cat("Hi, my name is ", students3$name, " and my roll number is ", students3$rollNo, ".", "\n", sep =
  cat("I have", students3$marksInMath, "marks in Maths,", students3$marksInPhysics, "in Physics, and", 
}

# Percentage Function
percentage.StudentS3 <- function(students3){
  cat("Percentage of Marks of ", students3$name, ":", sep = "", "\n")
  cat("Math = ", students3$marksInMath, "%", sep = "", "\n")
  cat("Physics = ", students3$marksInPhysics, "%", sep = "", "\n")
  cat("Chemistry = ", students3$marksInChemistry, "%", sep = "", "\n")
  cat("Average = ", (students3$marksInMath + students3$marksInPhysics + students3$marksInChemistry)/3,
}

# Topper Function
topper.StudentS3 <- function(...){
  x <- data.frame(name = ...$name, mark = (...$marksInMath + ...$marksInPhysics + ...$marksInChemistry)/
  cat(x$name[which.max(x$mark)], ": ", max(x$mark), "%", sep = "")
}
# 3 Student Profiles
students3_1 <- MakeStudentS3("Mridul", 1, 100, 90, 95)
students3_2 <- MakeStudentS3("Jacob", 2, 80, 75, 30)
students3_3 <- MakeStudentS3("Lyana", 3, 5, 10, 90)

# Example of Functions
print.StudentS3(students3_1)
```

```
## Hi, my name is Mridul and my roll number is 1.
## I have 100 marks in Maths, 90 in Physics, and 95 in Chemistry.
```

```
percentage.StudentS3(students3_3)
```

```
## Percentage of Marks of Lyana:
## Math = 5%
## Physics = 10%
## Chemistry = 90%
## Average = 35%
```

```
topper.StudentS3(students3_3, students3_1, students3_2)
```

```
## Lyana: 35%
```

S4:

```
# Creating Student
StudentS4 <- setClass("StudentS4", slots = list(name = "character", rollNo = "numeric", mM = "numeric",
```

```
setGeneric("print", function(students4){
})
setGeneric("percentage", function(students4){
})
setGeneric("topper", function(...){
})
```

```
# Print Function
setMethod("print", "StudentS4", function(students4){
  cat("Hi, my name is ", students4@name, " and my roll number is ", students4@rollNo, ".", "\n", sep =
  cat("I have", students4@mM, "marks in Maths,", students4@mP, "in Physics, and", students4@mC, "in Chem
})

# Percentage Function
setMethod("percentage", "StudentS4", function(students4){
  cat("Percentage of Marks of ", students4@name, ":", sep = "", "\n")
  cat("Math = ", students4@mM, "%", sep = "", "\n")
  cat("Physics = ", students4@mP, "%", sep = "", "\n")
  cat("Chemistry = ", students4@mC, "%", sep = "", "\n")
  cat("Average = ", (students4@mM + students4@mP + students4@mC)/3, "%", sep = "", "\n")
})

# Topper Function
setMethod("topper", "StudentS4", function(...){
  list_of_objects = list(...)
  list_of_dataframes <- lapply(seq_along(list_of_objects), function(x) {
        return(data.frame(name = list_of_objects[[x]]@name, mark = (list_of_objects[[x]]@mM + list_of_
    })
    x <- do.call(rbind, list_of_dataframes)
    cat(x$name[which.max(x$mark)], ": ", max(x$mark), "%", sep = "")
})

# 3 Student Profiles
students4_1 <- new("StudentS4", name = "Mridul", rollNo = 1, mM = 100, mP = 90, mC = 95)
students4_2 <- new("StudentS4", name = "Jacob", rollNo = 2, mM = 80, mP = 75, mC = 30)
students4_3 <- new("StudentS4", name = "Lyana", rollNo = 3, mM = 5, mP = 10, mC = 90)

# Example of Functions
print(students4_1)
```

```
## Hi, my name is Mridul and my roll number is 1.
## I have 100 marks in Maths, 90 in Physics, and 95 in Chemistry.
```

```
percentage(students4_3)
```

```
## Percentage of Marks of Lyana:
## Math = 5%
## Physics = 10%
## Chemistry = 90%
## Average = 35%
```

```
topper(students4_1, students4_2, students4_3)
```

```
## Mridul: 95%
```

Pros of single object:

- Easier to filter and search
- Easier for retrieving information
- Requires less memory (On a larger scale it'd be too large though)
- Clear and well-structured (though it does increase complexity of data management)

Pros of multiple objects:

- More specific data retrieval
- Easier to update specific information

Exercise 2: Recursive functions

Fibonacci Sequence:

```
fibonacci_sequence <- c(0, 1)

recursive.fibonacci <- function(n){
  if(n <= length(fibonacci_sequence)){
    return(fibonacci_sequence[n])
  } else{
    result <- recursive.fibonacci(n-1) + recursive.fibonacci(n-2)
    fibonacci_sequence <<- c(fibonacci_sequence, result)
    return(result)
  }
}

recursive.fibonacci(20)
```

```
## [1] 4181
```

Decimal Numbers -> Binary Number:

```
recursive.binary <- function(n)
  if(n == 0){
    return(0)
  } else if(n == 1){
    return(1)
  } else{
    return(cat(recursive.binary(floor(n/2)), n%%2))
  }

recursive.binary(75)
```

```
## 1 0 0 1 0 1 1
```

Exercise 3: Efficient Looping in R

For-Loop:

```
m1 <- matrix(c(1:9), nrow = 3, ncol = 3)
m2 <- matrix(c(9:1), nrow = 3, ncol = 3)

matrix_multiplier <- function(x, y){
  z <- matrix(, nrow = 3, ncol = 3)
  for(i in 1:3){
    for(j in 1:3){
      z[i, j] = 0
      for(k in 1:3){
        z[i, j] = z[i, j] + (x[i, k] * y[k, j])
      }
    }
  }
  return(z)
}

m3 <- matrix_multiplier(m1, m2)
m3
```

```
##      [,1] [,2] [,3]
## [1,]   90   54   18
## [2,]  114   69   24
## [3,]  138   84   30
```

Shapes:

```
a1 <- function(n){
  for(i in 1:n){
    cat("* * * * * * *", "\n")
  }
}

a1(10)
```

```
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
## * * * * * * *
```

```
b1 <- function(n){
  m = 1
  for(i in 1:n){
    for(j in 1:m){
      cat("* ")
    }
    m = m + 1
    cat("\n")
  }
}
```

```r
b1(9)
```

```
## *
## * *
## * * *
## * * * *
## * * * * *
## * * * * * *
## * * * * * * *
## * * * * * * * *
## * * * * * * * * *
```

```r
c1 <- function(n){
  for(i in 1:n){
    cat("* ")
  }
  cat("\n")
  for(i in seq(1, n-2, by = 1)){
    for(j in 1:i){
      cat(" ")
    }
    cat("*")

    for(k in 1:((2*n)-5-((i-1)*2))){
      cat(" ")
    }
    cat("*")
    cat("\n")
  }

  for(i in 1:(n-1)){
    cat(" ")
  }
  cat("*")
}

c1(11)
```

```
## * * * * * * * * * * *
##  *                 *
##   *               *
##    *             *
##     *           *
##      *         *
##       *       *
##        *     *
##         *   *
##          * *
##           *
```

```r
d1 <- function(n) {
  if(n%%2 == 0){
    for (i in 1:(n/2-1)) {
      for (j in (((n/2-i)*2)+1):2) {
        cat(" ")
      }
      for (k in 1:(i*2-1)) {
        cat("* ")
      }
      cat("\n")
    }

    for (i in 1:(n-1)) {
      cat("* ")
    }
    cat("\n")
    for (i in 1:(n-1)) {
      cat("* ")
    }
    cat("\n")

    for (i in (n/2-1):1) {
      for (j in 2:(((n/2-i)*2)+1)) {
        cat(" ")
      }
      for (k in 1:(2*i-1)) {
        cat("* ")
      }
      cat("\n")
    }
  } else{
    for (i in 1:(ceiling(n/2)-1)) {
      for (j in (((ceiling(n/2)-i)*2)+1):2) {
        cat(" ")
      }
      for (k in 1:(i*2-1)) {
        cat("* ")
      }
      cat("\n")
    }

    for (i in 1:n) {
      cat("* ")
    }
    cat("\n")

    for (i in (ceiling(n/2)-1):1) {
      for (j in 2:(((ceiling(n/2)-i)*2)+1)) {
        cat(" ")
      }
      for (k in 1:(2*i-1)) {
        cat("* ")
      }
      cat("\n")
    }
  }
}
```

```
d1(9)
```

```
##         *
##       * * *
##     * * * * *
##   * * * * * * *
## * * * * * * * * *
##   * * * * * * *
##     * * * * *
##       * * *
##         *
```

```
d1(10)
```

```
##         *
##       * * *
##     * * * * *
##   * * * * * * *
## * * * * * * * * *
## * * * * * * * * *
##   * * * * * * *
##     * * * * *
##       * * *
##         *
```

Exercise 4: Logic Questions

1. Doors: All doors that are open are perfect squares because they have an odd number of divisors.

```
list <- rep("closed", 100)
for(i in 1:100){
  for(j in seq(i, 100, by = i)){
    if(list[j] == "closed"){
      list[j] = "open"
    } else if(list[j] == "open"){
      list[j] = "closed"
    }
  }
}
table(list)
```

```
## list
## closed   open
##     90     10
```

```
which(list == "open")
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

2. You have to split the 8 balls into 2 groups: one with 2 (a) and the other with 6 (b). You take group b and split it up evenly on the scale (1st measurement). If they weigh the same, you way group a to find the heavier ball (2nd measurement). If the 1st measurement shows that the balls weigh differently you take the heavier group, remove one ball and weigh the other 2 (2nd measurement). If they weigh the same, the heavier ball is the one you removed, otherwise the measurement would show which one is heavier.

3. No, it's 50-50.

4. Answer is 7: You first divide them into 5 groups each with 5 horses and race them. The fastest then race against each other and the fastest out of them is the fastest of them all. Then you take the 2nd and 3rd fastest from the second race and race them against the 3 fastest horses from the fastest horse's group (excluding the fastest horse of course).

Day 2: Tidyverse

Lesson Day 2

Notes

What is tidyverse? Tidyverse is a collection of R packages that are designed to make data manipulation, visualization, and analysis more efficient and user-friendly. Its core packages include dplyr for data manipulation, tidyr for data tidying, ggplot2 for data visualization, readr for reading data, tibble for creating and working with tibbles (modern data frames), purrr for functional programming, stringr for manipulating strings, and forcats for working with categorical data. Tidyverse is widely used by data scientists, statisticians, and analysts for data wrangling and exploration in R.

Dplyr Tutorial:

Cheat Sheets

dplyr
ggplot2
tidyr
readr
stringr
purrr
forcats

Day 3: Vectors and Matrices Part I