Midterm Review

Navin Souda

2023-02-14

Basic Data Structures

Things to know:

- Atomic vs. generic vectors
- Data frames vs. Matrices/arrays
- Data types
  - Primitives
  - Coercion
  - `NA` vs. `NULL` vs. `NaN`

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
```

```
## [1] "list"
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
typeof(as.matrix(df))
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
```

```
## [1] "list"
```

```r
typeof(as.matrix(df))
```

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

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
typeof(as.matrix(df))
class(as.matrix(df))
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
```

```
## [1] "list"
```

```r
typeof(as.matrix(df))
```

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

```r
class(as.matrix(df))
```

```
## [1] "matrix" "array"
```

Basic Data Structures Practice I

```
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
typeof(as.matrix(df))
class(as.matrix(df))

typeof(df$y)
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
```

```
## [1] "list"
```

```r
typeof(as.matrix(df))
```

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

```r
class(as.matrix(df))
```

```
## [1] "matrix" "array"
```

Basic Data Structures Practice II

```
typeof(df$y)
```

```
## [1] "logical"
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
typeof(as.matrix(df))
class(as.matrix(df))

typeof(df$y)
class(df["y"])
```

Basic Data Structures Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
typeof(df)
```

```
## [1] "list"
```

```r
typeof(as.matrix(df))
```

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

```r
class(as.matrix(df))
```

```
## [1] "matrix" "array"
```

Basic Data Structures Practice II

```
typeof(df$y)
```

```
## [1] "logical"
```

```
class(df["y"])
```

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

Subsetting

Things to know:

- Subsetting 1-D objects (lists/vectors)
  - [] vs. [[]] vs. $
- Subsetting 2+-D objects (matrices, arrays, data frames)
- Simplification
  - Data frames vs. matrices
  - Named vectors
  - Factors

Subsetting Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
```

Subsetting Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
```

```
## [1]  TRUE  TRUE FALSE
```

Subsetting Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
df[, "y"]
```

Subsetting Practice I

```
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
```

```
## [1]  TRUE  TRUE FALSE
```

```
df[, "y"]
```

```
## [1]  TRUE  TRUE FALSE
```

Subsetting Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
df[, "y"]
df[4]
```

Subsetting Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
```

```
## [1]  TRUE  TRUE FALSE
```

```r
df[, "y"]
```

```
## [1]  TRUE  TRUE FALSE
```

```r
df[4]
```

```
## Error in `[.data.frame`(df, 4): undefined columns selected
```

Subsetting Practice I

```
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
df[, "y"]
df[4]
as.matrix(df)[4]
```

Subsetting Practice I

```r
df <- data.frame(x = c(1, 2, 3), y = c(T, T, F), z = c("A", "B",
    "C"))
df$y
```

```
## [1]  TRUE  TRUE FALSE
```

```r
df[, "y"]
```

```
## [1]  TRUE  TRUE FALSE
```

```r
as.matrix(df)[4]
```

```
## [1] "TRUE"
```

Conditions, Loops I

Things to know:

- Logical operators
  - `&`, `|`, `!`, `xor()`
  - Vectorized vs. non-vectorized
- Comparison operators
- `if()` statements
  - Results with input length 0, 1 or > 1
- Loops
  - `for()`, `while()`, `repeat`

Environments and scoping I

Things to know:

- Function basics
  - Arguments and default values
  - Scope of variables in function
  - `return()`
- Binding, copy-on-modify
- Environments
  - What environments do
  - Global environment and its parent
  - Base environment and its parent
  - Empty environment
- Function Environments
  - Enclosing environment
  - Calling environment
  - Execution environment

Environments and scoping II

    - Default behavior when searching for variables
    - `get()`
- Packages
    - Package environment vs. namespace
- Super assignment (`<<-`)

Environments and Scoping Practice I

```r
x <- "a"
y <- "b"
z <- "c"
f <- function(x, y, z) {
    z <- "d"
    print(g())  ## print here
    c(x, y, z)
}
g <- function(x = "p", y = "q") {
    c(x, y, get("z", envir = parent.frame()))
}
print(f(x, y, z))
```

Environments and Scoping Practice I

```r
x <- "a"
y <- "b"
z <- "c"
f <- function(x, y, z) {
    z <- "d"
    print(g())  ## print here
    c(x, y, z)
}
g <- function(x = "p", y = "q") {
    c(x, y, get("z", envir = parent.frame()))
}
print(f(x, y, z))
```

```
## [1] "p" "q" "d"
## [1] "a" "b" "d"
```

Environments and Scoping Practice I

```r
x <- "a"
y <- "b"
z <- "c"
f <- function(x, y, z) {
    z <- "d"
    print(g())  ## print here
    c(x, y, z)
}
g <- function(x = "p", y = "q") {
    z <<- "m"
    c(x, y, z)
}
paste(f(x, y, z), z)
```

Environments and Scoping Practice I

```r
x <- "a"
y <- "b"
z <- "c"
f <- function(x, y, z) {
    z <- "d"
    print(g())  ## print here
    c(x, y, z)
}
g <- function(x = "p", y = "q") {
    z <<- "m"
    c(x, y, z)
}
paste(c(f(x, y, z), z))
```

```
## [1] "p" "q" "m"
```

Environments and Scoping Practice II

```
## [1] "a" "b" "d" "m"
```

tidyr I

Things to know:

- Tidy data
  - Role of variables, observations, and values
- Pivoting
  - `pivot_wider()` vs. `pivot_longer()`

tidyr Practice I

```r
dat <- tibble(yr = rep(2011:2013, each = 3), ID = rep(c("A",
    "B", "C"), 3), score = sample(100, 9))
dat
```

```
## # A tibble: 9 x 3
##      yr ID    score
##   <int> <chr> <int>
## 1  2011 A        42
## 2  2011 B        17
## 3  2011 C        33
## 4  2012 A        88
## 5  2012 B        41
## 6  2012 C        11
## 7  2013 A        37
## 8  2013 B        22
```

tidyr Practice II

```
## 9  2013 C        94
```

```
pivot_wider(dat, names_from = "ID", values_from = "score")
```

tidyr Practice I

```r
dat <- tibble(yr = rep(2011:2013, each = 3), ID = rep(c("A",
    "B", "C"), 3), score = sample(100, 9))
dat
```

```
## # A tibble: 9 x 3
##      yr ID    score
##   <int> <chr> <int>
## 1  2011 A        89
## 2  2011 B        30
## 3  2011 C        64
## 4  2012 A        17
## 5  2012 B        75
## 6  2012 C        71
## 7  2013 A        60
## 8  2013 B        85
```

tidyr Practice II

```
## 9  2013 C         58
```

```
pivot_wider(dat, names_from = "ID", values_from = "score")
```

```
## # A tibble: 3 x 4
##      yr     A     B     C
##   <int> <int> <int> <int>
## 1  2011    89    30    64
## 2  2012    17    75    71
## 3  2013    60    85    58
```

dplyr I

Things to know:

- Main `dplyr` functions and chaining them together
  - `select()`, `filter()`, `mutate()`, `arrange()`, `summarise()`, `group_by()`
  - Joins

Regular Expressions I

Things to know:

- String functions
  - `str_c()`, `str_length()`, `str_sub()`, `str_trim()`, etc.
- Regex functions
  - `str_detect()`
  - `str_locate()`
  - `str_extract()`
  - `str_match()`
  - `str_split()`
- Regex patterns
  - Wildcards
  - Metacharacters
  - Character sets and classes
  - Anchors
  - Quantifiers
  - Capture groups
  - Lookarounds

Regular Expressions Practice I

```r
test_str <- "The quick. Brown fox. Jumps over. The lazy dog."
pattern1 <- "T.*."

str_match(test_str, pattern1)
```

Regular Expressions Practice I

```r
test_str <- "The quick. Brown fox. Jumps over. The lazy dog."
pattern1 <- "T.*."

str_match(test_str, pattern1)
```

```
##      [,1]
## [1,] "The quick. Brown fox. Jumps over. The lazy dog."
```

Regular Expressions Practice I

```r
test_str <- "The quick. Brown fox. Jumps over. The lazy dog."
pattern2 <- "T.*\\."

str_match(test_str, pattern2)
```

Regular Expressions Practice I

```r
test_str <- "The quick. Brown fox. Jumps over. The lazy dog."
pattern2 <- "T.*\\."

str_match(test_str, pattern2)
```

```
##      [,1]
## [1,] "The quick. Brown fox. Jumps over. The lazy dog."
```

Regular Expressions Practice I

```r
test_str <- "The quick. Brown fox. Jumps over. The lazy dog."
pattern3 <- "T.*?\\."

str_match(test_str, pattern3)
```

Regular Expressions Practice I

```r
test_str <- "The quick. Brown fox. Jumps over. The lazy dog."
pattern3 <- "T.*?\\."

str_match(test_str, pattern3)
```

```
##      [,1]
## [1,] "The quick."
```

S3 I

- Base types
- Generic functions
- Creating class-specific versions of generics
- Method dispatch
- Creating structures and giving objects classes

S3 Practice I

```r
showContents <- function(obj) {
    print(obj)
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    paste(obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

S3 Practice I

```r
showContents <- function(obj) {
    print(obj)
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    paste(obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

```
## $fruit
## [1] "Banana" "Apple"  "Pear"
##
## $quantity
## [1] 5 2 3
##
## attr(,"class")
## [1] "fruitbowl"
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

```
## [1] "Banana : 5" "Apple : 2"  "Pear : 3"
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
notebook <- structure(list(subjects = c("STATS 102A", "STATS 100C",
    "STATS 101A")))
showContents(notebook)
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

```
## [1] "Banana : 5" "Apple : 2"  "Pear : 3"
```

```r
notebook <- structure(list(subjects = c("STATS 102A", "STATS 100C",
    "STATS 101A")))
showContents(notebook)
```

```
## Error in UseMethod("showContents"): no applicable method for 'showConte
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
notebook <- structure(list(subjects = c("STATS 102A", "STATS 100C",
    "STATS 101A")))
showContents(notebook)
class(notebook) <- "notebook"
showContents(notebook)
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

```
## [1] "Banana : 5" "Apple : 2"  "Pear : 3"
```

```r
notebook <- structure(list(subjects = c("STATS 102A", "STATS 100C",
    "STATS 101A")))
showContents(notebook)
```

```
## Error in UseMethod("showContents"): no applicable method for 'showConte
```

```r
class(notebook) <- "notebook"
showContents(notebook)
```

```
## Subjects:  STATS 102A STATS 100C STATS 101A
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
notebook <- structure(list(subjects = c("STATS 102A", "STATS 100C",
    "STATS 101A")))
showContents(notebook)
class(notebook) <- "notebook"
showContents(notebook)
showContents.car(notebook)
```

S3 Practice I

```r
showContents <- function(obj) {
    UseMethod("showContents")
}
showContents.fruitbowl <- function(obj) {
    paste(obj$fruit, ":", obj$quantity)
}
showContents.car <- function(obj) {
    paste("Number of passengers:", obj$quantity)
}
showContents.notebook <- function(obj) {
    cat("Subjects: ", obj$subjects)
}
```

S3 Practice II

```r
obj1 <- structure(list(fruit = c("Banana", "Apple", "Pear"),
    quantity = c(5, 2, 3)), class = "fruitbowl")
showContents(obj1)
```

```
## [1] "Banana : 5" "Apple : 2"  "Pear : 3"
```

```r
notebook <- structure(list(subjects = c("STATS 102A", "STATS 100C",
    "STATS 101A")))
showContents(notebook)
```

```
## Error in UseMethod("showContents"): no applicable method for 'showConte
```

```r
class(notebook) <- "notebook"
showContents(notebook)
```

S3 Practice III

```
## Subjects:  STATS 102A STATS 100C STATS 101A
```

```r
showContents.car(notebook)
```

```
## [1] "Number of passengers: "
```

R6 I

- Creating new classes
  - `R6Class()` and its arguments
- Functions in classes and `invisible(self)`
- Modifying object creation with `initialize`
  - Changing a class does not affect existing objects with that class!!!
- Defining custom `print()`
- `set()`
- Inheritance
- Reference Semantics
  - Modification of object vs. copy-on-modify
  - Cloning

R6 Practice I

```r
library(R6)
Clock <- R6Class(classname = "Clock", public = list(hour = 0,
    minute = 0, second = 0, add_hour = function(x) {
        self$hour <- (self$hour + 1)%%24
    }, add_minute = function(x) {
        self$minute <- (self$minute + 1)%%60
    }, add_second = function(x) {
        self$second <- (self$second + 1)%%60
    }))
```

```r
clock1 <- Clock$new()
clock1$add_hour()$add_hour()$add_second()
```

R6 Practice I

```r
library(R6)
Clock <- R6Class(classname = "Clock", public = list(hour = 0,
    minute = 0, second = 0, add_hour = function(x) {
        self$hour <- (self$hour + 1)%%24
    }, add_minute = function(x) {
        self$minute <- (self$minute + 1)%%60
    }, add_second = function(x) {
        self$second <- (self$second + 1)%%60
    }))
```

```r
clock1 <- Clock$new()
clock1$add_hour()$add_hour()$add_second()
```

```
## Error in clock1$add_hour()$add_hour: $ operator is invalid for atomic ve
```

R6 Practice I

```r
library(R6)
Clock <- R6Class(classname = "Clock", public = list(hour = 0,
    minute = 0, second = 0, add_hour = function(x) {
        self$hour <- (self$hour + 1)%%24
        invisible(self)
    }, add_minute = function(x) {
        self$minute <- (self$minute + 1)%%60
        invisible(self)
    }, add_second = function(x) {
        self$second <- (self$second + 1)%%60
        invisible(self)
    }, print = function(...) {
        cat("Hour:", self$hour, "\n")
        cat("Minute:", self$minute, "\n")
        cat("Second:", self$second, "\n")
        invisible(self)
    }))
```

R6 Practice II

```r
clock1 <- Clock$new()
clock1$add_hour()$add_hour()$add_second()
clock1
```

R6 Practice I

```r
library(R6)
Clock <- R6Class(classname = "Clock", public = list(hour = 0,
    minute = 0, second = 0, add_hour = function(x) {
        self$hour <- (self$hour + 1)%%24
        invisible(self)
    }, add_minute = function(x) {
        self$minute <- (self$minute + 1)%%60
        invisible(self)
    }, add_second = function(x) {
        self$second <- (self$second + 1)%%60
        invisible(self)
    }, print = function(...) {
        cat("Hour:", self$hour, "\n")
        cat("Minute:", self$minute, "\n")
        cat("Second:", self$second, "\n")
        invisible(self)
    }))
```

R6 Practice II

```r
clock1 <- Clock$new()
clock1$add_hour()$add_hour()$add_second()
clock1
```

```
## Hour: 2
## Minute: 0
## Second: 1
```

R6 Practice I

```r
library(R6)
Clock <- R6Class(classname = "Clock", public = list(hour = 0,
    minute = 0, second = 0, add_hour = function(x) {
        self$hour <- (self$hour + 1)%%24
        invisible(self)
    }, add_minute = function(x) {
        self$minute <- (self$minute + 1)%%60
        invisible(self)
    }, add_second = function(x) {
        self$second <- (self$second + 1)%%60
        invisible(self)
    }, print = function(...) {
        cat("Hour:", self$hour, "\n")
        cat("Minute:", self$minute, "\n")
        cat("Second:", self$second, "\n")
        invisible(self)
    }))
```

R6 Practice II

```r
clock1 <- Clock$new()
clock1$add_hour()$add_hour()$add_second()
clock1
Clock$set("public", "clock_type", "analog")
clock1
```

R6 Practice I

```r
library(R6)
Clock <- R6Class(classname = "Clock", public = list(hour = 0,
    minute = 0, second = 0, add_hour = function(x) {
        self$hour <- (self$hour + 1)%%24
        invisible(self)
    }, add_minute = function(x) {
        self$minute <- (self$minute + 1)%%60
        invisible(self)
    }, add_second = function(x) {
        self$second <- (self$second + 1)%%60
        invisible(self)
    }, print = function(...) {
        cat("Hour:", self$hour, "\n")
        cat("Minute:", self$minute, "\n")
        cat("Second:", self$second, "\n")
        invisible(self)
    }))
```

R6 Practice II

```r
clock1 <- Clock$new()
clock1$add_hour()$add_hour()$add_second()
clock1
```

```
## Hour: 2
## Minute: 0
## Second: 1
```

```r
Clock$set("public", "clock_type", "analog")
clock1
```

```
## Hour: 2
## Minute: 0
## Second: 1
```