Challenge-3

Elise Wong

2023-08-28

I. Questions

Question 1: Emoji Expressions

Imagine you're analyzing social media posts for sentiment analysis. If you were to create a variable named "postSentiment" to store the sentiment of a post using emojis (😄 for positive, 😐 for neutral, 😢 for negative), what data type would you assign to this variable? Why?

Solution: I would assign this variable the 'character' data type. An emoji is a categorical nominal variable - it is non-numeric and does not have intrinsic ordering to the categories. Since a single emoji can be considered a single character, similar to that of a letter of the alphabet e.g. 'A', 'B', it also falls under the 'single' data type under the 'character' group.

Question 2: Hashtag Havoc

In a study on trending hashtags, you want to store the list of hashtags associated with a post. What data type would you choose for the variable "postHashtags"? How might this data type help you analyze and categorize the hashtags later?

Solution: I would choose the 'character' data type, under specifically the 'string' category. This is as hashtags are a categorical nominal variable - they can be considered strings of alphabets and/or numerals. This data may help me be more flexible in handling multiple hashtags associated with a single post; allow me to group hashtags by topic, sentiment, popularity, or any other criteria; and more.

Question 3: Time Traveler's Log

You're examining the timing of user interactions on a website. Would you use a numeric or non-numeric data type to represent the timestamp of each interaction? Explain your choice.

Solution: I would use a numeric data type to represent the exact timestamp and/or interval of each interaction. In particular, I would choose that of type 'double'. This is as timestamps are fundamentally numerical values that represent a specific point in time, usually measured as the number of seconds or milliseconds from a specific reference point.

Alternatively, use a function specifically for date/time, such as as.Date().

Question 4: Event Elegance

You're managing an event database that includes the date and time of each session. What data type(s) would you use to represent the session date and time?

Solution: I would use a 'character - string' data type to represent the session date, and use a 'double' data type to represent the session time.

Question 5: Nominee Nominations

You're analyzing nominations for an online award. Each participant can nominate multiple candidates. What data type would be suitable for storing the list of nominated candidates for each participant?

Solution: The ‘character - string’ data type would be most suitable for storing the list of nominated candidates for each participant.

Question 6: Communication Channels

In a survey about preferred communication channels, respondents choose from options like “email,” “phone,” or “social media.” What data type would you assign to the variable “preferredChannel”?

Solution: I would assign the variable the ‘character - string’ data type.

Question 7: Colorful Commentary

In a design feedback survey, participants are asked to describe their feelings about a website using color names (e.g., “warm red,” “cool blue”). What data type would you choose for the variable “feedbackColor”?

Solution: I would choose the ‘character - string’ data type for the variable.

Question 8: Variable Exploration

Imagine you’re conducting a study on social media usage. Identify three variables related to this study, and specify their data types in R. Classify each variable as either numeric or non-numeric.

Solution: Here, three variables can include but are not limited to: number of posts (numeric), age of user (numeric), and platform used (non-numeric).

Question 9: Vector Variety

Create a numeric vector named “ages” containing the ages of five people: 25, 30, 22, 28, and 33. Print the vector.

Solution:

```r
# Load tidyverse package,
library("tidyverse")
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ─
─
## ✔ dplyr     1.1.0     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.1.8
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ─
─
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force a
ll conflicts to become errors
```

```r
# Create a numeric vector,
ages <- c(25, 30, 22, 28, 33)
ages
```

```
## [1] 25 30 22 28 33
```

Question 10: List Logic

Construct a list named “student_info” that contains the following elements:

- A character vector of student names: “Alice,” “Bob,” “Catherine”
- A numeric vector of their respective scores: 85, 92, 78
- A logical vector indicating if they passed the exam: TRUE, TRUE, FALSE

Print the list.

Solution:

```
# To construct the list,
student_info = list(
  studentnames = c("Alice", "Bob", "Catherine"),
  scores = c(85, 92, 78),
  passedexam = c(TRUE, TRUE, FALSE)
)

# To print the list,
student_info
```

```
## $studentnames
## [1] "Alice"     "Bob"       "Catherine"
##
## $scores
## [1] 85 92 78
##
## $passedexam
## [1]  TRUE  TRUE FALSE
```

Question 11: Type Tracking

You have a vector “data” containing the values 10, 15.5, “20”, and TRUE. Determine the data types of each element using the typeof() function.

Solution:

```
# To find data type of element '10',
data <- c(10, 15.5, "20", TRUE)

# To determine the data types of each element,
typeof(data[1])
```

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

```
typeof(data[2])
```

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

```
typeof(data[3])
```

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

```
typeof(data[4])
```

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

```
# This is an instance of implicit coercion.
```

Question 12: Coercion Chronicles

You have a numeric vector “prices” with values 20.5, 15, and “25”. Use explicit coercion to convert the last element to a numeric data type. Print the updated vector.

Solution:

```
# Create the vector,
prices <- c(20.5, 15, "25")

# Using explicit coersion to convert last element to numeric data type,
prices <- as.numeric(prices)

# Hence, to view the updated vector and verify its numeric status,
prices
```

```
## [1] 20.5 15.0 25.0
```

```
typeof(prices)
```

```
## [1] "double"
```

Question 13: Implicit Intuition

Combine the numeric vector c(5, 10, 15) with the character vector c(“apple”, “banana”, “cherry”). What happens to the data types of the combined vector? Explain the concept of implicit coercion.

Solution:

```
# To combine vectors,
numeric <- c(5, 10, 15)
character <- c("apple", "banana", "cherry")
combinedvector <- c(numeric, character)
combinedvector
```

```
## [1] "5"      "10"     "15"     "apple"  "banana" "cherry"
```

```
# To assess data type of combined vector,
typeof(combinedvector)
```

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

```
# The resulting vector falls under the 'character' data type. In all, the concept o
f implicit coercion can be understood in two parts - coercion is the automatic conv
ersion by R from one type to another, and the implicit nature stems from how R conv
erts the type based on its contents to accommodate all element types present in the
relevant vector.
```

Question 14: Coercion Challenges

You have a vector “numbers” with values 7, 12.5, and “15.7”. Calculate the sum of these numbers. Will R automatically handle the data type conversion? If not, how would you handle it?

Solution:

```
# Create a vector,
numbers <- c(7, 12.5, "15.7")

# R will not automatically handle the data type conversion, as "15.7" is a characte
r with a decimal point. As such, to calculate the sum,
newnumbers <- c(7, 12.5, as.numeric("15.7"))
sum(newnumbers)
```

```
## [1] 35.2
```

```
# The sum of the vector is 35.2.
```

Question 15: Coercion Consequences

Suppose you want to calculate the average of a vector “grades” with values 85, 90.5, and “75.2”. If you directly calculate the mean using the mean() function, what result do you expect? How might you ensure accurate calculation?

Solution:

```
# Create a vector,
grades <- c(85, 90.5, "75.2")

# If I directly calculate the mean using the mean() function, I expect to get an er
ror message as R does not automatically convert character values with decimals into
numeric values within a vector.

# As such, I will use explicit coercion,
gradesnew <- c(85, 90.5, as.numeric("75.2"))
mean(gradesnew)
```

```
## [1] 83.56667
```

```
# The mean is 83.6 marks (3 s.f.).
```

Question 16: Data Diversity in Lists

Create a list named “mixed_data” with the following components:

- A numeric vector: 10, 20, 30
- A character vector: “red”, “green”, “blue”
- A logical vector: TRUE, FALSE, TRUE

Calculate the mean of the numeric vector within the list.

Solution:

```
# Create a list,
mixed_data = list(
  numeric = c(10, 20, 30),
  character = c("red", "green", "blue"),
  logical = c(TRUE, FALSE, TRUE)
)

# Print the list,
mixed_data
```

```
## $numeric
## [1] 10 20 30
##
## $character
## [1] "red"   "green" "blue"
##
## $logical
## [1]  TRUE FALSE  TRUE
```

```
# Calculate the mean of the numeric vector within the list,
mean(mixed_data$numeric)
```

```
## [1] 20
```

```
# The mean within the list is 20.
```

Question 17: List Logic Follow-up

Using the “student_info” list from Question 10, extract and print the score of the student named “Bob.”

Solution:

```
# To extract and print the score of the student named "Bob",
bobscore <- student_info$scores[student_info$studentnames == "Bob"]
bobscore
```

```
## [1] 92
```

```
# Bob scored 92 marks on the test.
```

Question 18: Dynamic Access

Create a numeric vector 'values' with random values. Write R code to dynamically access and print the last element of the vector, regardless of its length.

Solution:

```
# Create a numeric vector,
values <- sample(1:100, 10)
values
```

```
##  [1] 19 54 66 22 34 96 94 98 73 58
```

```
# To print the last element of the vector,
values[10]
```

```
## [1] 58
```

```
# Alternatively, use tail(),
tail(values, n = 1)
```

```
## [1] 58
```

Question 19: Multiple Matches

You have a character vector words <- c("apple", "banana", "cherry", "apple"). Write R code to find and print the indices of all occurrences of the word "apple."

Solution:

```
# Create the vector,
words <- c("apple", "banana", "cherry", "apple")

# To find the occurrences of the word "apple",
words[words=="apple"]
```

```
## [1] "apple" "apple"
```

```
# To print the indices of all occurrences of the word "apple",
indices_apple <- which(words=="apple")
indices_apple
```

```
## [1] 1 4
```

Question 20: Conditional Capture

Assume you have a vector ages containing the ages of individuals. Write R code to extract and print the ages of individuals who are older than 30.

Solution:

```
# Create the vector,
ages <- c(sample(0:100, 20))

# To extract and print the ages of individuals older than 30,
ages[ages > 30]
```

```
##  [1] 73 71 35 55 63 92 68 51 84 56 98 88 99 31
```

Question 21: Extract Every Nth

Given a numeric vector sequence <- 1:20, write R code to extract and print every third element of the vector.

Solution:

```
# Create the vector,
sequence <- 1:20

# To extract and print every third element of the vector,
everythird <- sequence[seq(from = 1, to = 20, by = 3)]
everythird
```

```
## [1]  1  4  7 10 13 16 19
```

Question 22: Range Retrieval

Create a numeric vector numbers with values from 1 to 10. Write R code to extract and print the values between the fourth and eighth elements.

Solution:

```
# Create the vector,
numbers <- 1:10

# To extract the values,
numbers[4:8]
```

```
## [1] 4 5 6 7 8
```

Question 23: Missing Matters

Suppose you have a numeric vector data <- c(10, NA, 15, 20). Write R code to check if the second element of the vector is missing (NA).

Solution:

```
# Create the vector,
data <- c(10, NA, 15, 20)

# To isolate the second element of the vector,
data[2]
```

```
## [1] NA
```

```
# To check if the second element (or any element) of the vector is missing,
if(is.na(data[2])) {
  print("The second element of the vector is missing (NA).")
} else {
  print("The second element of the vector is not missing.")
}
```

```
## [1] "The second element of the vector is missing (NA)."
```

Question 24: Temperature Extremes

Assume you have a numeric vector temperatures with daily temperatures. Create a logical vector hot_days that flags days with temperatures above 90 degrees Fahrenheit. Print the total number of hot days.

Solution:

```
# Create a numeric vector 'temperatures' with daily temperatures in Fahrenheit,
temperatures <- c(sample(50:150, 10))
temperatures
```

```
##  [1]  63 135  86  56  54 124  55  95 113  98
```

```
# Create a logical vector hot_days,
hot_days <- temperatures > 90

# To print the total number of hot days,
sum(hot_days)
```

```
## [1] 5
```

Question 25: String Selection

Given a character vector fruits containing fruit names, create a logical vector long_names that identifies fruits with names longer than 6 characters. Print the long fruit names.

Solution:

```
# Create a character vector 'fruits' containing fruit names,
fruits <- c("apple", "banana", "watermelon", "cherry", "apple", "strawberry")

# To identify the long fruit names,
long_names <- nchar(fruits) > 6
long_names
```

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

```
# To print the long fruit names,
fruits[long_names]
```

```
## [1] "watermelon" "strawberry"
```

Question 26: Data Divisibility

Given a numeric vector numbers, create a logical vector divisible_by_5 to indicate numbers that are divisible by 5. Print the numbers that satisfy this condition.

Solution:

```
# Create a numeric vector numbers,
numbers <- c(sample(1:100, 10))
numbers
```

```
##  [1] 27 81 95  7 80 10 43  4 29 85
```

```
# Create a logical vector to indicate numbers that are divisible by 5,
divisible_by_5 <- numbers %% 5 == 0
divisible_by_5
```

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

```
# Print numbers that satisfy condition,
numbers[divisible_by_5]
```

```
## [1] 95 80 10 85
```

Question 27: Bigger or Smaller?

You have two numeric vectors vector1 and vector2. Create a logical vector comparison to indicate whether each element in vector1 is greater than the corresponding element in vector2. Print the comparison results.

Solution:

```r
# Create two numeric vectors vector1 and vector2,
vector1 <- c(sample(1:100,10))
vector1
```

```
##  [1] 98 96 85 21 45 53 15  4 19 78
```

```r
vector2 <- c(sample(1:100,10))
vector2
```

```
##  [1] 44  3 77 69  7 72 16 96 79 52
```

```r
# To compare,
comparison <- vector1 > vector2

# To print the results,
comparison
```

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

Thank you for your time!