3.5

# Before we continue, make sure you know

# what objects you have in your environment.

# Use a function to get a list of the object in your environment ????

ls()

A close up of a number

Description automatically generated

# Create a variable called lvl that contains the following values: 8, 10, 10, 1, 10, 10, 8, 12, 1, 12.

lvl <- c( 8, 10, 10, 1, 10, 10, 8, 12, 1, 12)



# Use the commands listed below to do the following operations:

#Find the sum of the elements in lvl

Sum(lvl)



#Find the average of the elements in lvl



#Find the median value (the middle value) of the elements in lvl



#Get R to return the length of the lvl variable



#Find the standard deviation of the values in lvl



#Find the standard deviation of the elements in lvl, then round it; do it in a single command



#Find the standard deviation of the elements in lvl, then round it, then specifically ask R to print it; do it in a single command



#Functions to use: sum(), mean(), median(), length(), sd(), round(), print()

# Tip: when typing code in the console and you want to run different commands on the same object, use the Up Arrow to access the last command you executed, and simply edit the function call, keeping the rest the same.

3.6

# Find what arguments the sample() function takes. We will need that for later.

args(sample)

A black text on a white background

Description automatically generated

# Find out what arguments the median function takes. While calling the median for the lvl object, try setting the na.rm = argument to TRUE.

A close up of a logo

Description automatically generated



# Does R return a different value from the one when na.rm = is set to FALSE (NB: the default value for the na.rm = argument is FALSE; this means that whenever we do not explicitly specify that it must be set to TRUE, it is set to FALSE)?

No . return same value , because the lvl not contain NA

# Now try the same command but with the atk variable.





# Is the output different? Why do you think that is?

Yes, it is. Due to the atk does contains NA

3.7

# This is the code for the card-drawing function we did in the lesson:

draw <- function(){

deck <- c("Duke", "Assassin", "Captain", "Ambassador", "Contessa")

hand <- sample(deck, size=3, replace = T)

print(hand)

}

# Following the schema laid out there, write a function that flips a coin 100 times.

A screenshot of a computer code

Description automatically generated

# Hint: create a coin object that stores the values "heads" and "tails".

# Make the coin unfair so that 70% of the time it comes up tails.

A black text on a white background

Description automatically generated

A computer code with blue text

Description automatically generated

# Hint: recall the arguments that the sample() function takes; use the prob = argument to make the coin unfair. Use the combine function to pass to it the values 0.3 (30%) and 0.7 (70%).

4.3

# Imagine that you are working for the Ghostbusters,

# and for safety reasons you are recording the height and weight

# for everyone or everything the team encounters.

# So far you have collected information about the following.

# entities: Dr Peter Venkman, Dr Raymond Stantz, Dr. Eagon Spengler, Dana Barrett, Vigo, Slimer, and the Marshmallow Man.

enitties <- c("Dr Peter Venkman", "Dr Raymond Stantz","Dr. Eagon Spengler", "Dana Barrett","Vigo", "Slimer", "the Marshmallow Man")

# Create the following two vectors:

# A vector called weight, storing the values: 71, 67, 83, 67

Weight <- c(71, 67, 83, 67)



# A vector called height, storing the values: 1.75, 1.81, 1.78, 1.82, 1.97, 2.12, 2.75



# You need to calculate the BMI for all you have data for. BMI is calculated by dividing the weight in kg by the height in meters squared.

# Carry out the operation and save the result in a variable called bmi.

bmi <- weight/(height\*height)



# What is your output?

A white background with red text

Description automatically generated

# Try printing your bmi variable.

A number on a white background

Description automatically generated

# Can you explain how you got the last three values, given that your weight variable only had 4 elements?

The last three values are calculated by dividing the last value in the weight vector (67) by each of the last three values in the height vector (1.97, 2.12, and 2.75). Since the input vectors are not the same length, R will recycle the values in the weight vector to match the length of the height vector. Specifically, R will repeat the weight vector until it has the same length as the height vector, and then perform the element-wise division.  
weight will come that to calculate c(71, 67, 83, 67, 71, 67, 83)

4.4

# Requirements: the following vectors: the original cards object,

# and atk from the second exercise.

cards <- c("Blue-Eyes White Dragon", "Exodius", "The Winged Dragon of Ra", "Raigeki", "Slifer the Sky Dragon", "Obelisk the Tormentor", "Black Luster Soldier", "5-Headed Dragon", "Exodia the Forbidden One", "Dragon Master Knight")

atk <- c(3000, NA, NA, NA, NA, 4000, 3000, 5000, 1000, 5000)

# Assign a name to each element from the atk vector that corresponds to a character from the cards vector.



# Check the attributes of the atk element.

A black text on a white background

Description automatically generated

# Is there another command you can use to check if the atk object has names assigned to it?

A black text on a white background

Description automatically generated

# Remove the names.



# Can you think of another way to assign names to the atk object?



4.6

# Requirements: the following vectors: the named atk object from the previous exercise.

cards <- c("Blue-Eyes White Dragon", "Exodius", "The Winged Dragon of Ra", "Raigeki", "Slifer the Sky Dragon", "Obelisk the Tormentor", "Black Luster Soldier", "5-Headed Dragon", "Exodia the Forbidden One", "Dragon Master Knight")

atk <- c(3000, NA, NA, NA, NA, 4000, 3000, 5000, 1000, 5000)

# Extract the following values from the atk object:

#The 6-th value (can you do it in more than one way)

atk[6]

A black and white text

Description automatically generated

A close up of a sign

Description automatically generated

#All the values but the 2nd one

A white background with black text

Description automatically generated

#Values 1, 3, 5, 7, and 9

A white background with black text

Description automatically generated

#All the values but the 4th, 5th, and 6th

A white background with black text

Description automatically generated

#All the values larger than 2000

A close-up of a sign

Description automatically generated

But they are has NA, to not include NA

A close up of words

Description automatically generated

4.7

# Create a vector s that stores the number

# sequence 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30.

# Use the seq() command to do that.

s <- seq(2, 30, by = 2)





# Check the attributes of s.





# Does it have any dimensions?



s variable is a one-dimensional vector in R, so it does not have any dimensions other than its length

# Give it dimensions. Try specifying 3 rows and 5 columns.



A screenshot of a computer

Description automatically generated

# What happens if we try to go beyond the traditional height and width properties and specify a value for depth, too. Let's create a hypercube that has 1 row, 3 columns and 5 "slices".





A white background with black numbers

Description automatically generated

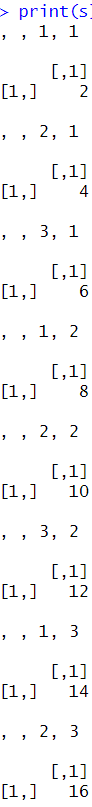
# Try assigning the following dimensions to s: 1, 3, 5.

dim(s) <- c(1,3,5)

# Can you do more than that? Do you think this could be useful for something?







5.2

# Pictured below is a matrix.

# Recreate this matrix in at least three different ways.

#Player Piece

#[1,] "dark" "king"

#[2,] "dark" "queen"

#[3,] "dark" "pawn"

#[4,] "dark" "pawn"

#[5,] "dark" "knight"

#[6,] "light" "bishop"

#[7,] "light" "king"

#[8,] "light" "rook"

#[9,] "light" "pawn"

#[10,] "light" "pawn"

# Hint. Use the dim() function, the matrix() function, and cbind().

Player <- c("dark","dark","dark","dark","dark","light","light","light","light","light")

piece <- c("king", "queen", "pawn", "pawn", "knight", "bishop", "king", "rook", "pawn", "pawn")

A screenshot of a computer program

Description automatically generated

chess2 <- c("dark","dark","dark","dark","dark","light","light","light","light","light","king", "queen", "pawn", "pawn", "knight", "bishop", "king", "rook", "pawn", "pawn")

A computer screen shot of a computer code

Description automatically generated

rawData <- c("dark","dark","dark","dark","dark","light","light","light","light","light","king", "queen", "pawn", "pawn", "knight", "bishop", "king", "rook", "pawn", "pawn")

chess3 <- matrix(data =rawData, ncol = 2)

A screen shot of a computer code

Description automatically generated

# Can you think of a faster way to create this matrix?

Use cbind is fast way

# Assign names to it in two different ways.

A screenshot of a computer code

Description automatically generated

A computer code with black text

Description automatically generated

# Would using rbind() be practical in this case?

# Yes

# Create vectors for player and piece information

player <- c("dark", "dark", "dark", "dark", "dark", "light", "light", "light", "light", "light")

piece <- c("king", "queen", "pawn", "pawn", "knight", "bishop", "king", "rook", "pawn", "pawn")

# Create 10 small vectors, one for each row of the matrix

row1 <- c(player[1], piece[1])

row2 <- c(player[2], piece[2])

row3 <- c(player[3], piece[3])

row4 <- c(player[4], piece[4])

row5 <- c(player[5], piece[5])

row6 <- c(player[6], piece[6])

row7 <- c(player[7], piece[7])

row8 <- c(player[8], piece[8])

row9 <- c(player[9], piece[9])

row10 <- c(player[10], piece[10])

# Combine the 10 small vectors using rbind() to create the matrix

chess\_matrix <- rbind(row1, row2, row3, row4, row5, row6, row7, row8, row9, row10)

print(chess\_matrix)

A screenshot of a computer code

Description automatically generated

# Can you think of a way to make adding a row to the matrix worthwhile?

Yes, there are several reasons why adding a row to the matrix can be worthwhile. Here are a few examples:

Updating the matrix with new game information: In a game of chess, as the game progresses, new pieces may be added to the board. By adding a new row to the matrix for each new piece, we can keep track of the state of the game and analyze it later.

Analyzing the performance of individual players: By adding a new row to the matrix for each game played by a player, we can analyze their performance over time. For example, we can calculate their win-loss record or their average number of pieces captured per game.

Recording data over time: If the matrix represents data collected over time, adding a new row for each time point can be useful for tracking changes and trends in the data.

Predicting future outcomes: By analyzing patterns in the data, we can use the matrix to make predictions about future outcomes. Adding a new row to the matrix can provide more data points for our analysis and improve the accuracy of our predictions.

Overall, adding a row to the matrix can be worthwhile if it provides additional information that is useful for analysis, prediction, or decision-making.