Name: Xian Hui B. Cheng
Section: BSIT 3-5

Activity 5: Using Built-in Functions and Control Structures

Explore R's built-in functions and control structures. Provide R code and a brief explanation for each task. Test your solutions in an R environment.

1. Given a vector of names c("alice", "BOB", "Charlie", "DIANA"), write code to: Convert all names to title case (e.g., "Alice", "Bob"). Add a prefix "ID_" to each name, resulting in names like "ID_Alice".

```
name <- c("alice", "BOB", "Charlie", "DIANA")
titleCase_name<- tools::toTitleCase(tolower(names))
prefixed_name <- paste0("ID_", titleCase_name)
print(prefixed_name)
```

Output:

```
> print(prefixed_name)
[1] "ID_Alice"   "ID_Bob"     "ID_Charlie" "ID_Diana"
```

Explanation:

tolower() - this converts all the characters in the argument into lowercase. This is needed before we convert it to titlecase to ensure that all the letters have no capital word

tools::toTitleCase() - this converts each word to title case(Sample Word) where the first letter is always capitalized

paste0("ID_", titleCase_name) - this concatenates each word without the spaces into “ID_” (ID_+name)

2. Write an R function that takes a numeric vector as input and returns a list with the following: Mean, Median, Standard Deviation. Test this function with a random vector of 20 values generated using rnorm().

```r
calculate<- function(vec) {
  mean_value <- mean(vec)
  median_value <- median(vec)
  sd_value <- sd(vec)

  return(list(
    Mean = mean_value,
    Median = median_value,
    Standard_Deviation = sd_value
  ))
}

random_num<- rnorm(20)
result <- calculate(random_num)
print(result)
```

Output:

```
$Mean
[1] -0.05125716

$Median
[1] -0.1399433

$Standard_Deviation
[1] 0.8299387
```

Explanation:

calculate function

- This calculates the mean, median, and standard deviation using the functions mean(), median(), and sd() and returns all the values into list.

random_num

- This creates a vector random_num with 20 random numbers sampled from a standard normal distribution (mean = 0, standard deviation = 1).

calculate(random_num)

- This calls the calculate function and passes the random_num vector as input. The result is a list containing the mean, median, and standard deviation of the random numbers.

3. Create a script that takes a numeric vector and classifies each number into: "Positive" if greater than 0, "Negative" if less than 0, "Zero" otherwise. Ensure the script handles edge cases (e.g., empty vector).

```
classify_numbers <- function(numbers) {
  if (length(numbers) == 0) {
    return("The input is empty.")
  }

  classifications <- sapply(numbers, function(number) {
    if (number > 0) {
      return("Positive")
    } else if (number < 0) {
      return("Negative")
    } else {
      return("Zero")
    }
  })

  return(classifications)
}
```

Sample input:

```
input_numbers <- c(-2, 0, 3, -5, 7)
print(classify_numbers(input_numbers))

empty_list <- numeric(0)
print(classify_numbers(empty_list))

zero_list <- c(0, 0, 0)
print(classify_numbers(zero_list))
```

Output:

```
> print(classify_numbers(input_numbers))
[1] "Negative" "Zero"     "Positive" "Negative" "Positive"
>
> empty_list <- numeric(0)
> print(classify_numbers(empty_list))
[1] "The input is empty."
>
> zero_list <- c(0, 0, 0)
> print(classify_numbers(zero_list))
[1] "Zero" "Zero" "Zero"
```

Explanation:

length(numbers) == 0

- this checks if the input numbers is empty (length(numbers) == 0), the function returns a message "The input is empty." immediately and also handles if no data to process

classifications <- sapply(numbers, function(number)

The sapply() function applies the logic to each number in the numbers list.
Inside the anonymous function:

- Numbers greater than 0 are classified as "Positive".
- Numbers less than 0 are classified as "Negative".
- Numbers equal to 0 are classified as "Zero".

4. Given a vector of scores c(70, 85, 90, 65, 95, 88), write code to calculate the average score only for scores above 80.

```
scores <- c(70, 85, 90, 65, 95, 88)
scores_above_80 <- scores[scores > 80]
average<- mean(scores_above_80)

print(average)
```

Output:

```
[1] 89.5
```

Explanation:

scores[scores > 80]:

- Filters the scores vector to include only values greater than 80. This creates a new vector scores_above_80.

Calculating the Mean:

- mean(scores_above_80): Computes the average of the filtered scores.

5. Write a recursive R function to compute the nth Fibonacci number. Demonstrate the function with inputs n = 5 and n = 10.

```
fibonacci <- function(n) {
  if (n <= 0) {
    return(0)
  } else if (n == 1) {
    return(1)
  } else {
    return(fibonacci(n - 1) + fibonacci(n - 2))
  }
}
```

```r
n5_result <- fibonacci(5)
n10_result <- fibonacci(10)

# Print the results
print(paste("Fibonacci number (n = 5): ", n5_result))
print(paste("Fibonacci number (n = 10): ", n10_result))
```

Output:

```
> print(paste("Fibonacci number (n = 5): ", n5_result))
[1] "Fibonacci number (n = 5):  5"
> print(paste("Fibonacci number (n = 10): ", n10_result))
[1] "Fibonacci number (n = 10):  55"
>
```

Explanation:

fibonacci function

Logic (in fibonacci):

In fibonacci, the nth of a Fibonacci number is the sum of the two preceding Fibonacci numbers.

F(0)=0: When n≤ 0, the Fibonacci number is defined as 0.
F(1) = 1: When n = 1. The Fibonacci number is 1

In the recursive case: For n>1n > 1n>1, the function computes F(n)=F(n−1)+F(n−2). This means the the nth Fibonacci number is the