1st Statistical Inference Assignment

Juan Pablo Silvestre

2024-08-29

Q1

Factorials

Given a non-negative integer, return its factorial value.

```r
factorial = function(n){
    if (n == 0 || n == 1){
      return (1)
    }
    else {
      return( n * factorial(n - 1))
    }
  }

data = c(5, 7, 10)

for(i in 1:length(data)){
  print(paste(data[i], "=", factorial(data[i])))
}
```

```
## [1] "5 = 120"
## [1] "7 = 5040"
## [1] "10 = 3628800"
```

For this question we're using the recursive factorial formula, which denotes: $n! = n \times (n-1)!$. First, we check the first case, where n is equal to 0 or 1. By definition $0! = 1$, which can be proven by plugin 1 for n:

The recursive formula for factorial is given by:

$$n! = n \times (n-1)!$$

For $n = 1$:

$$1! = 1 \times 0!$$

And by definition:

$$0! = 1$$

Upon base case review, the function calls itself with the argument $n-1$, multiplying the result by n, this goes on until it reaches the base case.

Q2

Even and Odd Numbers

Given an integer, return whether its even or odd.

```r
is_even = function(x){
  if (x %% 2 == 0){
    return ("is even!")
  }
  else {
    return ("is odd!")
  }
}


data = c(sample(1:100, 4))

for( i in 1:length(data)){
  print(paste(data[i], is_even(data[i])))
}
```

```
## [1] "3 is odd!"
## [1] "99 is odd!"
## [1] "56 is even!"
## [1] "72 is even!"
```

This a simple question to solve, by using the module (%%) operator, which returns the remainder of a division, we can check if the input number is even if the remainder of module 2 is equal to 0.

Q3

Vowel Or Consonant?

Given a matrix of random letters, return "V" (Vowel) or "C" (Consonant) for each respective letter.

```r
letter_to_evaluate = function(letter){

  vowels = c("a", "e", "i", "o", "u")

  if (letter %in% vowels){
    return ("V")
  }
  else {
    return ("C")
  }
}

my_letters = matrix(sample(letters, 9, replace = FALSE), nrow = 3)

evaluated_letter = matrix(apply(my_letters, c(1, 2), letter_to_evaluate), ncol = 3)

print(evaluated_letter)
```

```
##      [,1] [,2] [,3]
## [1,] "C"  "C"  "C"
## [2,] "C"  "C"  "V"
## [3,] "C"  "C"  "C"
```

The solution involves a vector that contains all the vowels.

We first create our matrix with random non-repeated letters using the R built-in constant "letters", that contains the lowercase version from 'a' to 'z'. Next, we declare the function that evaluates if the letter is present in the "vowels" vector, returning "V" or "C" upon evaluation. Finally, we create our output matrix, which content is filled by applying the function across its elements, this is achieved using the built-in function "apply" that takes 3 arguments:

- X: an array, including a matrix.
- MARGIN: A vector giving the subscripts which the function will be applied over. For a matrix 'c(1, 2)' indicates rows and columns.
- FUN: the function to be applied.

Q4

Vowel or Consonant? Part 2

Given a vector containing the names of the group members (in this case myself), return the evaluated name by applying the function from the previous question.

```
evaluate_name = function(name){

  ev_name = c()
  name_letters = unlist(strsplit(name, split = ""))

  for (i in 1:length(name_letters)){
    ev_name = c(ev_name, letter_to_evaluate(name_letters[i]))
  }

  return (paste(ev_name, collapse = "-"))
}

data = c("juan", "pablo", "silvestre", "gonzalez")

for (i in 1:length(data)){
  print(paste(data[i], ":", evaluate_name(data[i])))
}
```

```
## [1] "juan : C-V-V-C"
## [1] "pablo : C-V-C-C-V"
## [1] "silvestre : C-V-C-C-V-C-C-C-V"
## [1] "gonzalez : C-V-C-C-V-C-V-C"
```

This solution is not different from the other, but it has some tweaks to deal with, such as splitting the string names and printing the correct output format.

First, we declare our function that will iterate through each string, splitting it into its individual letters, then we recall our "letter_to_evaluate" function, analyzing and appending each letter to the "ev_name" vector.

The desired format is a sequence of letters concatenated with a separator between each letter, for this particular case, the separator used is "-". Using the built-in "paste" function and passing "collapse" as one of the parameters, we are able to obtain the output format.

Q5

Multipliaction Table

Given a integer, return its multiplication table.

```r
mult_table  = function(n){

  for (i in 1:10){
    print(paste(i, "x", n, "=", i * n))
  }
}

mult_table(3)
```

```
## [1] "1 x 3 = 3"
## [1] "2 x 3 = 6"
## [1] "3 x 3 = 9"
## [1] "4 x 3 = 12"
## [1] "5 x 3 = 15"
## [1] "6 x 3 = 18"
## [1] "7 x 3 = 21"
## [1] "8 x 3 = 24"
## [1] "9 x 3 = 27"
## [1] "10 x 3 = 30"
```

Simple problem involving a simple solution, we declare the function that takes our number as parameter, then it iterates from 1 to 10, multiplying and printing the corresponding values.

Q6

Approximation of e

The number e can be approximated using the following formula:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

Where n is a positive integer.

We are assigned to design a program that computes the value of e using this iterative formula, where our last iteration is defined by this condition.

$$|e_{n-1} - e_n| \leq 10^{-7}$$

Where e_{n-1} and e_n are the values computed on the $n-1$ and n respectively.

```r
compute_e = function(err){

  e_old = 1

  for (i in 1:50){

    term = 1 / factorial(i)
    e_new = e_old + term

    if (abs(e_old - e_new) < err){
      break
    }
    else{
      e_old = e_new
    }

    print(paste(i, ":", e_new))
  }
}

err = 10^(-7)
compute_e(err)
```

```
## [1] "1 : 2"
## [1] "2 : 2.5"
## [1] "3 : 2.66666666666667"
## [1] "4 : 2.70833333333333"
## [1] "5 : 2.71666666666667"
## [1] "6 : 2.71805555555556"
## [1] "7 : 2.71825396825397"
## [1] "8 : 2.71827876984127"
## [1] "9 : 2.71828152557319"
## [1] "10 : 2.71828180114638"
```

For this approach, we implement a for loop, that iterates until it reaches 50 if the given condition does not satisfy, in this case the error value is relatively small, so 50 iterations is enough.

Here, we declare e_{n-1} (e_old) as 1 for our $n-1$ iteration, then we dive into our for loop that adds up the "term" variable to the e_n (e_new), aside from e_{n-1} which is updated to e_n on every iteration the condition does not satisfy.

Note: Any feedback regarding my redaction is appreciated.