Worksheet-1 in R

Worksheet for R Programming

Instructions:

- Use RStudio or the RStudio Cloud accomplish this worksheet.
- Create an .RMD file and name the file as *RWorksheet_lastname#1.Rmd*. Knit the rmd file into a pdf, save it as *RWorksheet_lastname#1.pdf*
- Create your own *GitHub repository* and push the R script as well as this pdf worksheet to your own repo (see Unit 2).

Accomplish this worksheet by answering the questions being asked and writing the code manually.

Using functions:

`seq()`, `assign()`, `min()`, `max()`, `c()`, `sort()`, `sum()`, `filter()`

1. Set up a vector named age, consisting of `34, 28, 22, 36, 27, 18, 52, 39, 42, 29, 35, 31, 27, 22, 37, 34, 19, 20, 57, 49, 50, 37, 46, 25, 17, 37, 42, 53, 41, 51, 35, 24, 33, 41`.

a. How many data points?

34 Data points

b. Write the R code and its output.

```
length_age <- length(age)
print(length_age)
```

```
[1] 34
```

2. Find the reciprocal of the values for age
Write the R code 1 and its output.

```
reciprocal_age <- 1 / age
print(reciprocal_age)
```

```
[1] 0.02941176 0.03571429 0.04545455 0.02777778 0.03703704 0.05555556 0.01923077 0.02564103
 [9] 0.02380952 0.03448276 0.02857143 0.03225806 0.03703704 0.04545455 0.02702703 0.02941176
[17] 0.05263158 0.05000000 0.01754386 0.02040816 0.02000000 0.02702703 0.02173913 0.04000000
[25] 0.05882353 0.02702703 0.02380952 0.01886792 0.02439024 0.01960784 0.02857143 0.04166667
[33] 0.03030303 0.02439024
```

3. Assign also `new_age <- c(age, 0, age)`.

What happen to the new_age?
The length of new_age double the original length of age.

4. Sort the values for age.

Write the R code and its output.

```
sorted_age <- sort(age)
print(sorted_age)
```

```
[1] 17 18 19 20 22 22 24 25 27 27 28 29 31 33 34 34 35 35 36 37 37 37 39 41 41 42 42 46 49 50 51
[32] 52 53 57
```

5. Find the minimum and maximum value for age.

Write the R code and its output.

```
min_age <- min(age)
max_age <- max(age)
print(min_age)
print(max_age)
```

```
> print(min_age)
[1] 17
> print(max_age)
[1] 57
```

6. Set up a vector named data, consisting of `2.4, 2.8, 2.1, 2.5, 2.4, 2.2, 2.5,2.3, 2.5, 2.3, 2.4, and 2.7`.

a. How many data points?
12 Data points

b. Write the R code and its output.

```
data <- c(2.4, 2.8, 2.1, 2.5, 2.4, 2.2, 2.5, 2.3, 2.5, 2.3, 2.4, 2.7)

length_data <- length(data)
print(length_data)
```

```
[1] 12
```

7. Generates a new vector for data where you double every value of the data. | What happen to the data?
The data vector reamins unchanged. While, new vector doubled_data is created, where each value is doubled compared to the original data.

8. Generate a sequence for the following scenario:

8.1 Integers from `1 to 100`.

```
seq_1_to_100 <- seq(1, 100)
print (seq_1_to_100)
```

```
 [1]   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[24]  24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
[47]  47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
[70]  70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
[93]  93 94 95 96 97 98 99 100
```

8.2 Numbers from `20 to 60`

```
seq_20_to_60 <- seq(20, 60)
print (seq_20_to_60)
```

```
[1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
[32] 51 52 53 54 55 56 57 58 59 60
```

*8.3 Mean of numbers from `20 to 60`

```
mean_20_to_60 <- mean(20:60)
print (mean_20_to_60)
```

```
[1] 40
```

*8.4 Sum of numbers from `51 to 91`

```
sum_51_to_91 <- sum(51:91)
print (sum_51_to_91)
```

```
[1] 2911
```

*8.5 Integers from `1 to 1,000`

```
seq_1 <- seq(1, 1000)
print (seq_1)
```

a. How many data points from 8.1 to 8.4?

143 Data points

b. Write the R code and its output from 8.1 to 8.4.

```
total_data_points <- length_1_to_100 + length_20_to_60 + length_mean + length_sum
print(total_data_points)
```

c. For 8.5 find only maximum data points until 10.

```
> max_until_10 <- max(seq_1_to_1000[1:10])
> print(max_until_10)
[1] 10
```

9.*Print a vector with the integers between 1 and 100 that are not divisible by 3 5 and7 using filter option.

```
Filter(function(i) { all(i %% c(3,5,7) != 0) }, seq(100))
```

Write the R code and its output.

```
filtered_numbers <- Filter(function(i) { all(i %% c(3, 5, 7) != 0) }, seq(100))
print(filtered_numbers)
```

```
[1]  1  2  4  8 11 13 16 17 19 22 23 26 29 31 32 34 37 38 41 43 44 46 47 52 53 58 59 61 62 64 67
[32] 68 71 73 74 76 79 82 83 86 88 89 92 94 97
```

10. Generate a sequence backwards of the integers from `1 to 100`.

Write the R code and its output.

```
seq_backwards <- seq(100, 1)
print(seq_backwards)
```

```
[1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83  82  81  80  79  78
[24]  77  76  75  74  73  72  71  70  69  68  67  66  65  64  63  62  61  60  59  58  57  56  55
[47]  54  53  52  51  50  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32
[70]  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10   9
[93]   8   7   6   5   4   3   2   1
```

11. List all the natural numbers below 25 that are multiples of 3 or 5.Find the sum of these multiples.

```
> print(multiples_3_or_5)
 [1]  3  5  6  9 10 12 15 18 20 21 24
> print(sum_multiples)
[1] 143
```

a. How many data points from 10 to 11? **112 Data points**

b. Write the R code and its output from 10 and 11.

```
total_data_points <- length(seq_backwards) + length(multiples_3_or_5) + length(sum_multiples)
print(total_data_points)
```

```
[1] 112
```

12. Statements can be grouped together using braces '`{`' and '`}`'. A group of statements is sometimes called a **block**. Single statements are evaluated when a new line is typed at the end of the syntactically complete statement. Blocks are not evaluated until a new line is entered after the closing brace.

Enter this statement:

```
x <- {0 + x + 5 + }
```

Describe the output.

Error: unexpected '}' in "x <- {0 + x + 5 + }"

Error occurs because the expression 0 + x + 5 + is incomplete.

13. *Set up a vector named score, consisting of `72, 86, 92, 63, 88, 89, 91, 92, 75, 75` and `77`. To access individual elements of an atomic vector, one generally uses the `x[i]` construction. Find `x[2]` and `x[3]`. Write the R code and its output.

```
score <- c(72, 86, 92, 63, 88, 89, 91, 92, 75, 75, 77)

x2 <- score[2]
x3 <- score[3]
print(x2)
print(x3)
```

```
> print(x2)
[1] 86
> print(x3)
[1] 92
```

14. *Create a vector `a = c(1,2,NA,4,NA,6,7)`.

a. Change the NA to 999 using the codes `print(a,na.print="-999")`.

```
a <- c(1, 2, 999, 4, 999, 6, 7)
```

b. Write the R code and its output. Describe the output.

```
a <- c(1, 2, 999, 4, 999, 6, 7)
print(a, na.print = "-999")
```

```
[1]   1   2 999   4 999   6   7
```

First it displays -999 where the NA values were, without changing the underlying data. When I change the NA to 999 it displays 999 where the NA values were.

15. A special type of function calls can appear on the left hand side of the assignment operator as in > `class(x) <- "foo"`.

Follow the codes below:

```
name = readline(prompt="Input your name: ")
age = readline(prompt="Input your age: ")
print(paste("My name is",name, "and I am",age ,"years old."))
print(R.version.string)
```

What is the output of the above code?

> name =readline(prompt="Inputyourname:")

Inputyourname:John

> age =readline(prompt="Inputyourage:")

Inputyourage:19

> print(paste("Mynameis",name,"andIam",age,"yearsold."))

[1] "Mynameis John andIam 19 yearsold."

> print(R.version.string)

[1] "R version 4.4.1 (2024-06-14)"