RWorksheet.Sorenio#1.Rmd

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

   b. Write the R code and its output.
   ```
   age <- c(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)
   print(age)

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

2. Find the reciprocal of the values for age.

   Write the R code and its output.

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

```
> reciprocalage <- 1 / age
>
> print(reciprocalage)
 [1] 0.02941176 0.03571429 0.04545455 0.02777778
 [5] 0.03703704 0.05555556 0.01923077 0.02564103
 [9] 0.02380952 0.03448276 0.02857143 0.03225806
[13] 0.03703704 0.04545455 0.02702703 0.02941176
[17] 0.05263158 0.05000000 0.01754386 0.02040816
[21] 0.02000000 0.02702703 0.02173913 0.04000000
[25] 0.05882353 0.02702703 0.02380952 0.01886792
[29] 0.02439024 0.01960784 0.02857143 0.04166667
[33] 0.03030303 0.02439024
```

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

```
new_age <- c(age, 0, age)
```

What happen to the new_age?

The new_age vector now contains all the elements of the age vector, followed by a single 0, and then all the elements of the age vector again.

4. Sort the values for age.

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

Write the R code and its output.

```
> print(age_sorted)
 [1] 17 18 19 20 22 22 24 25 27 27 28 29 31 33 34
[16] 34 35 35 36 37 37 37 39 41 41 42 42 46 49 50
[31] 51 52 53 57
```

5. Find the minimum and maximum value for age.

Write the R code and its output.

```
min_age <- min(age)
```

print(min_age)

max_age <- max(age)
print (max_age)

```
> min_age <- min(age)
>
> print(min_age)
[1] 17
>
> max_age <- max(age)
>
> 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

   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)

      numdatapoints <- length(data)
      print(numdatapoints)

```
> 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)
>
> # a. How many data points
> # b. Write the R code and its output.
>
> numdatapoints <- length(data)
>
> print(numdatapoints)
[1] 12
```

7. Generates a new vector for data where you double every value of the data. | What happen to the data?
   doubled_data <- data * 2

print(doubled_data)

The value in the data vector is multiplied by 2, where all the values are exactly twice their original values.

```
> doubled_data <- data * 2
>
> print(doubled_data)
 [1] 4.8 5.6 4.2 5.0 4.8 4.4 5.0 4.6 5.0 4.6 4.8
[12] 5.4
```

8. Generate a sequence for the following scenario:

8.1 Integers from 1 to 100.
int_1_to_100 <- 1:100

8.2 Numbers from 20 to 60
num_20_to_60 <- 20:60

*8.3 Mean of numbers from 20 to 60
mean_20_to_60 <- mean(numbers_20_to_60)

*8.4 Sum of numbers from 51 to 91
ints_1_to_1000 <- 1:1000

*8.5 Integers from 1 to 1,000
ints_1_to_1000 <- 1:1000

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.

```
variable8_1 <- length(int_1_to_100)
variable8_2 <- length(num_20_to_60)
variable8_3 <- 1
variable8_4 <- 1

totaldatapoints <- variable8_1 + variable8_2 + variable8_3 + variable8_4
print(totaldatapoints)
```

```
> variable8_1 <- length(int_1_to_100)
>
> variable8_2 <- length(num_20_to_60)
>
> variable8_3 <- 1
>
> variable8_4 <- 1
>
> totaldatapoints <- variable8_1 + variable8_2 + vari
able8_3 + variable8_4
>
> print(totaldatapoints)
[1] 143
```

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

```
ints_up_to_10 <-
ints_1_to_1000[ints_1_to_1000 <= 10]
maxvalue <- max(ints_up_to_10)
print(maxvalue)
```

```
> ints_up_to_10 <-
+
+ ints_1_to_1000[ints_1_to_1000 <= 10]
>
> maxvalue <- max(ints_up_to_10)
>
> print(maxvalue)
[1] 10
```

9. Print a vector with the integers between 1 and 100 that are not divisible by 3, 5 and 7 using filter option.

`Filter(function(i) { all(i %% c(3,5,7) != 0) }, seq(100))`
Write the R code and its output.

```
not_divisible <- Filter(function(i) { all(i %% c(3, 5, 7) != 0) }, seq(1, 100))
print(not_divisible)
```

```
> print(not_divisible)
 [1]  1  2  4  8 11 13 16 17 19 22 23 26 29 31 32
[16] 34 37 38 41 43 44 46 47 52 53 58 59 61 62 64
[31] 67 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.

```
backwards_sequence <- rev(1:100)
print(backwards_sequence)
```

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

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

Find the sum of these multiples.

```
multsof3to5 <- seq(1, 24)[seq(1, 24) %% 3 == 0 | seq(1, 24) %% 5 == 0]
 print(multsof3to5)
```

3 5 6 9 10 12 15 18 20 21 24

```
SumOfMultiples <- sum(multsof3to5)
SumOfMultiples
```

143

a. How many data points from 10 to 11?
There are 111 data points. In number 10, there are 100. In number 11, there are 11. 112 if we include the sum of multiples of 3 or 5.

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

```
backwards_sequence <- rev(1:100)
print(backwards_sequence)

multsof3to5 <- seq(1, 24)[seq(1, 24) %% 3 == 0 | seq(1, 24) %% 5 == 0]
```

print(multsof3to5)

SumOfMultiples <- sum(multsof3to5)
SumOfMultiples

numbers <- 1:24
multiples <- numbers[numbers %% 3 == 0 | numbers %% 5 == 0]

sum_of_multiples <- sum(multiples)
print(multiples)
print(sum_of_multiples)

```
> backwards_sequence <- rev(1:100)
> print(backwards_sequence)
  [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83  82  81
 [21]  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65  64  63  62  61
 [41]  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  44  43  42  41
 [61]  40  39  38  37  36  35  34  33  32  31  30  29  28  27  26  25  24  23  22  21
 [81]  20  19  18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1
>
> multsof3to5 <- seq(1, 24)[seq(1, 24) %% 3 == 0 | seq(1, 24) %% 5 == 0]
> print(multsof3to5)
 [1]  3  5  6  9 10 12 15 18 20 21 24
>
> SumOfMultiples <- sum(multsof3to5)
> SumOfMultiples
[1] 143
>
> numbers <- 1:24
> multiples <- numbers[numbers %% 3 == 0 | numbers %% 5 == 0]
>
> sum_of_multiples <- sum(multiples)
> print(multiples)
 [1]  3  5  6  9 10 12 15 18 20 21 24
>
> print(sum_of_multiples)
[1] 143
```

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.

It says: Error: unexpected '}' in 'x <- {0 + x + 5 + }'.

It results in a syntax error since the expression is clearly lacking. In order to run the code, there should be a value after the last '+' to complete the expression that is intended to be calculated.

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)
score_2 <- score[2]
score_3 <- score[3]
print(score_2)
print(score_3)
```

```
> score <- c(72, 86, 92, 63, 88, 89, 91, 92, 75, 75, 77)
>
> score_2 <- score[2]
>
> score_3 <- score[3]
>
> print(score_2)
[1] 86
>
> print(score_3)
[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").
    b. Write the R code and its output. Describe the output.

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

    In the output, the NA values are replaced by -999, while all other values remain the same.

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="Input your name: ")
Input your name: Althea Mae
> age = readline(prompt="Input your age: ")
Input your age: 18
> print(paste("My name is", name, "and I am", age, "years old."))
[1] "My name is Althea Mae and I am 18 years old."
> print(R.version.string)
[1] "R version 4.4.1 (2024-06-14)"
```

The output is letting the user input data and creates message based on the user's input, followed by the R version used.