Assignment 1

Reagan Anderson

2024-10-02

Exercise 1:

Create a vector of three elements (2,4,6) and name that vector `vec_a`. Create a second vector, `vec_b`, that contains (8,10,12). Add these two vectors together and name the result `vec_c`.

```
vec_a <- c(2,4,6)
vec_b <- c(8,10,12)
vec_c <- c(vec_a, vec_b)
```

Exercise 2:

Create a vector, named `vec_d`, that contains only two elements (14,20). Add this vector to `vec_a`. What is the result and what do you think R did (look up the recycling rule using Google)? What is the warning message that R gives you?

```
vec_d <- c(14,20)
vec_e <- vec_a + vec_d
```

```
## Warning in vec_a + vec_d: longer object length is not a multiple of shorter
## object length
```

Answer: The result is a vector containing the elements (2,4,6,14,20). R did use the recycling rule. The warning is that the "object length is not a multiple of shorter object length" meaning that object of vec_a and vec_d are not the same lengths, and therefore the function of recycling the values repeats until all elements of vec_a have been added to.

Exercise 3:

Next add 5 to the vector vec_a. What is the result and what did R do? Why doesn't it give you a warning message similar to what you saw in the previous problem?

```
vec_a <- vec_a + 5
```

Answer: The result is a vector contianing the elements (2,4,6,5). It doesn't give a warning because 5 is not a vector, and therefore only has the size of 1 and is added to every element of vec_a.

Exercise 5:

Generate the vector of even numbers
$$\{2,4,6,\dots,20\}$$
a) Using the seq() function and

```
seq(2,20,2)
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

b) Using the a:b shortcut and some subsequent algebra. *Hint: Generate the vector 1-10 and then multiple it by 2.*

```
seq(1:10) * 2
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

Exercise 6:

Generate a vector of 21 elements that are evenly placed between 0 and 1 using the `seq()` command and name this vector `x`.

```
x <- seq(0,1, length.out=21)
x
```

```
##  [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70
## [16] 0.75 0.80 0.85 0.90 0.95 1.00
```

Exercise 8:

Generate the vector
$$\{2,2,2,2,4,4,4,4,8,8,8,8\}$$
using the `rep()` command. You might need to check the help file for rep() to see all of the options that rep() will accept. In particular, look at the optional argument `each=`.

```
# c(rep(2,4), rep(4,4), rep(8,4))
# rep(c(2,4,8), each=4)
rep(2:8, c(4,0,4,0,0,0,4))
```

```
##  [1] 2 2 2 2 4 4 4 4 8 8 8 8
```

Exercise 11:

Create and manipulate a data frame.

a) Create a `data.frame` named `my.trees` that has the following columns:

- Girth = {8.3, 8.6, 8.8, 10.5, 10.7, 10.8, 11.0}
- Height= {70, 65, 63, 72, 81, 83, 66}
- Volume= {10.3, 10.3, 10.2, 16.4, 18.8, 19.7, 15.6}

```r
my.trees <- data.frame(
  Girth = c(8.3, 8.6, 8.8, 10.5, 10.7, 10.8, 11.0),
    Height = c(70, 65, 63, 72, 81, 83, 66),
    Volume = c(10.3, 10.3, 10.2, 16.4, 18.8, 19.7, 15.6)
)
my.trees
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
## 7  11.0     66   15.6
```

b) Without using **`dplyr`** functions, extract the third observation (i.e. the third row)

```r
my.trees[3,]
```

```
##   Girth Height Volume
## 3   8.8     63   10.2
```

c) Without using **`dplyr`** functions, extract the Girth column referring to it by name (don't use whatever order you placed the columns in).

```r
my.trees["Girth"]
```

```
##   Girth
## 1   8.3
## 2   8.6
## 3   8.8
## 4  10.5
## 5  10.7
## 6  10.8
## 7  11.0
```

d) Without using **`dplyr`** functions, print out a data frame of all the observations *except* for the fourth observation. (i.e. Remove the fourth observation/row.)

```r
my.trees[-4,]
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 5  10.7     81   18.8
## 6  10.8     83   19.7
## 7  11.0     66   15.6
```

e) Without using `dplyr` functions, use the `which()` command to create a vector of row indices that have a `girth` greater than 10. Call that vector `index`.

```
index <- which(my.trees$Girth > 10)
```

f) Without using `dplyr` functions, use the `index` vector to create a small data set with just the large girth trees.

```
trees_large <- data.frame(
  my.trees[index,]
)
trees_large
```

```
##   Girth Height Volume
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
## 7  11.0     66   15.6
```

```
# slice(my.trees, index)
```

g) Without using `dplyr` functions, use the `index` vector to create a small data set with just the small girth trees

```
trees_small <- data.frame(
  my.trees[-index,]
)
trees_small
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
```

```
# slice(my.trees, -index)
```

Exercise 12:

The following code creates a `data.frame` and then has two different methods for removing the rows with `NA` values in the column `Grade`. Explain the difference between the two.

```
    df <- data.frame(name= c('Alice','Bob','Charlie','Daniel'),
                     Grade = c(6,8,NA,9))

    df[ -which(  is.na(df$Grade) ), ]
    df[  which( !is.na(df$Grade) ), ]
```

Answer: The first example takes the column 'Grades' and finds the 'NA' values and returns true using the is.na() function. Then takes the not true and removes them from the 'df' data frame using the which() with a '-' at the beginning to represent the not true. The second example takes the same column 'Grades' and finds the is 'NA' values, using the is.na() function. Then returns the not true, because of the '!' in front of the is.na to show the not, and then removing the false values from the data frame.

Exercise 14:

Create and manipulate a list. a) Create a list named my.test with elements: + x = c(4,5,6,7,8,9,10) + y = c(34,35,41,40,45,47,51) + slope = 2.82 + p.value = 0.000131

```
x <- c(4,5,6,7,8,9,10)
y <- c(34,35,41,40,45,47,51)
slope <- 2.82
p.value <- 0.000131

my_list <- list(X_values = x, Y_values = y, Slope = slope, P_value = p.value)
str(my_list)
```

```
## List of 4
##  $ X_values: num [1:7] 4 5 6 7 8 9 10
##  $ Y_values: num [1:7] 34 35 41 40 45 47 51
##  $ Slope   : num 2.82
##  $ P_value : num 0.000131
```

b) Extract the second element in the list.

```
my_list[ 2 ]
```

```
## $Y_values
## [1] 34 35 41 40 45 47 51
```

c) Extract the element named `p.value` from the list.

```
my_list['P_value']
```

```
## $P_value
## [1] 0.000131
```