Week 1 Exercises

Ray Chandonnet

October 29, 2022

Please complete all exercises below WITHOUT using any libraries/packages.

Exercise 1

Assign 10 to the variable x. Assign 5 to the variable y. Assign 20 to the variable z.

```
x <- 10
y <- 5
z <- 20
print(x)
```

```
## [1] 10
```

```
print(y)
```

```
## [1] 5
```

```
print(z)
```

```
## [1] 20
```

Exercise 2

Show that x is less than z but greater than y.

Note: your output must be a SINGLE boolean, do not output a boolean for each expression.

```
Test1 <- x<z & x>y
print(Test1)
```

```
## [1] TRUE
```

Exercise 3

Show that x and y do not equal z.

Note: your output must be a SINGLE boolean, do not output a boolean for each expression.

```
Test2 <- x!=z & y!=z
print(Test2)
```

```
## [1] TRUE
```

Exercise 4

Show that the formula `x + 2y = z`.

Note: your output must be a SINGLE boolean

```
Test3 <- x+2*y == z
print(Test3)
```

```
## [1] TRUE
```

Exercise 5

I have created a vector (Test_vector) of integers for you. Determine if any of x, y, or z are in the vector.

Note: your output must be a SINGLE boolean, do not output a boolean for each expression.

```
Test_vector <- c(1,5,11:22)
x_found <- x %in% Test_vector
y_found <- y %in% Test_vector
z_found <- z %in% Test_vector
Any_found <- x_found | y_found | z_found
print(Any_found)
```

```
## [1] TRUE
```

Exercise 6

Show which value is contained in the test vector. To do this you will need to create an element-wise logical vector using operators. `x == vector`. Once you have done that you will need to use slicing to return all indices that have matches. **Note: your output should be two integers**

First, I create a vector containing the three values I am searching for (x, y, z)

```
Searchfor <- c(x,y,z)
```

Now: It seems there are at least two ways to do this. The first way I figured out is to create a logical vector "Founds" which is the same length as the test vector, and contains booleans indicating whether each value in Test_vector is in the Searchfor vector. Then you create a new vector that slices the test vector using those booleans. That code and results are below: (I am outputting the logical vector to demonstrate that it worked correctly)

```
Founds <- Test_vector %in% Searchfor
Common1 <- Test_vector[Founds]
print(Founds)
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12]  TRUE FALSE FALSE
```

```
print(Common1)
```

```
## [1]  5 20
```

However, I also discovered thanks to Professor Google that there is a vector function called "Intersect" that returns the same results without the two step process, which might reduce runtime speed if this was a real-life exercise with a very large vector. I include those results here - I believe that "intersect"" is a base function in R so it doesn't violate the "no libraries / packages" rule of the assignment :) I would be interested in knowing whether you prefer the approach you recommended or this approach? Code and Results are below:

```
Common2 <- intersect(Searchfor,Test_vector)
print(Common2)
```

```
## [1]  5 20
```

OK - after reading the example that was emailed, I tried to find a third way that used the "==" operator you presented in the example accompanying question 6. Sure, I can cycle through each variable and show whether it's in the test vector (using slicing for effect rather than the %in% function). But I don't know what that accomplishes, it seems brutish. The other way maybe to do it is to slice the testvector using the "==" operator instead of %in% operator. Interestingly, I get the correct results but with an error warning! So I'm going to stick with one of my first two solutions unless you tell me that there is a learning opportunity I missed in here somewhere involving the "==" operator!

```
print(Test_vector[x==Test_vector])
```

```
## numeric(0)
```

```
print(Test_vector[y==Test_vector])
```

```
## [1] 5
```

```
print(Test_vector[z==Test_vector])
```

```
## [1] 20
```

```
Common3 <- Test_vector[Searchfor == Test_vector]
```

```
## Warning in Searchfor == Test_vector: longer object length is not a multiple
## of shorter object length
```

```
print(Common3)
```

```
## [1]  5 20
```

One final note I would love your comments on: My original "programmer's instinct" was to create a Common vector with no values in it, and then run a For loop, looping through the values I'm searching for and dynamically adding each value to my "Common" if it is found in the test vector. Is this an inefficient way of doing this? Or is that in essence what the intersect function is doing "behind the scenes"? Thanks for indulging me!

I LIED. That was my thought when I submitted the second time yesterday. I thought that was my final note, but then I started researching the concept of "vectorization" today for the assignment this week and discovered that one of the benefits of vectorization (using vectors in operations, and doing the slicing stuff), is that running operatings on vectors makes your code more efficient (because the operation is done automatically on each element or combination of elements in the vector) versus manual "for loops"!!! Have I discovered that exact benefit in the way I attacked problem 6, simple problem though it might be? If so cool! (And that's part of the reason why I am making a third / final submission)