Bonnie Turek

Eco 634 – Lab 1: Fundamentals 2 (Lab 2?)

9/15/2021

Q1.

```
vec_2 <- vec_1 == 3
>vec_1[vec_2]
```

*According to the check above, the output is a vector of all 3s

Q2. Two reasons why determining which elements in vec_1 have value 3 by visual inspection is a bad idea

-This is very time consuming

-Due to human error you may miss a 3 and the results be inaccurate

Q3. We don't get the same count of 3 each time after running the code because the sample() function takes a random sample "of the specified size from the elements of x using either with or without replacement". Sample function is grabbing different random values each time. There is going to be a different number of 3s in each randomly generated matrix. Therefore, the sum of 3s depend on how many 3s are in the random sample vector. This can change each time.

Q4. Considering the different vectors generated each time, explain why using a logical test is a safe way to select entries with a value of 3:

Using a logical test is standalone code that will not need to change depending on the input vector. You can therefore reuse a logical test for different datasets and randomly generated vectors. The output of a logical test can in fact change, and this will depend on the contents of the input vector in question. Even though the different vectors generated each time will be populated with random numbers, the logical test will still be able to pull out the values of 3 each time. If you were performing a logical test by hand and selecting the values equal to 3 in each vector visually, then this would change with each vector and could be very time consuming.

Q5. Why is performing logical ‘by hand’ subsetting is very very bad practice? You may want consider re-usability of code, working with different sized data sets, and sharing code with collaborators.

Logical “by hand” subsetting I assume is visually picking out values in vectors that would be equal to 3. With very large datasets, this would be an extremely time consuming process and could introduce human error if a 3 is missed. If you use logical subsetting code, this could be shared code and would guarantee the same result each time, contingent upon the same input vector each time. If the logical subsetting code was shared with others, they would also be guaranteed to get the same result.

Q6. Basic loop modified:

```
for (i in 1:10)

{

  print(paste0("This is loop iteration:", i))

}
```

Output:

```
[1] "This is loop iteration:1"

[1] "This is loop iteration:2"

[1] "This is loop iteration:3"

[1] "This is loop iteration:4"

[1] "This is loop iteration:5"

[1] "This is loop iteration:6"

[1] "This is loop iteration:7"

[1] "This is loop iteration:8"

[1] "This is loop iteration:9"

[1] "This is loop iteration:10"
```

Q7. Run loop for n times:

```
n = 5

for (i in 1:n)

{

  print(i)

}
```

You can change the value of n and this code will run on your computer as well. N can be changed to any integer value. The for loop code will then execute the function body ‘n’ times. For example, you set n equal to 5, the for loop will run 5 times.

Q8. You should be able to open the below code in a new RStudio session and change the value of n if you choose – the code should still run but just be altered to the value of n you input. Vec_1 and n are both defined in the code below so that’s why the code is self-contained and will run on another laptop R session.

```
n = 3

vec_1 = sample(12, n, replace = TRUE)

for (i in 1:n)

{

  print(paste0("The element of vec_1 at index ", i, " is ", vec_1[i],"."))

}
```

Where vec_1 = 8 4 12   Output:

```
[1] "The element of vec_1 at index 1 is 8."

[1] "The element of vec_1 at index 2 is 4."

[1] "The element of vec_1 at index 3 is 12."
```

Q9.

```
create_and_print_vec = function(n, min = 1, max = 10)
{
  vec_new = sample(min:max, n, replace = TRUE)
  print(vec_new)
  for (i in 1:n)
  print(paste0("The element at index ", i, " is ", vec_new[i],"."))
}
create_and_print_vec(10, 100, 2000)
```

Above is the code for the create and print vector function that 1. Creates a vector of n random integers between the values of min and max and 2. Loops through the elements of the vector and prints a message with the index of the element and its value.

The output for above code:

```
(this is just vec_new printed out) [1]  451  889 1471  981 1125  835 1158 1980 1853  806
[1] "The element at index 1 is 451."
[1] "The element at index 2 is 889."
[1] "The element at index 3 is 1471."
[1] "The element at index 4 is 981."
[1] "The element at index 5 is 1125."
[1] "The element at index 6 is 835."
[1] "The element at index 7 is 1158."
[1] "The element at index 8 is 1980."
[1] "The element at index 9 is 1853."
[1] "The element at index 10 is 806."
```

The function runs the for loop 'n' number of times – we can enter any integer value for n (which is the first argument of our function) but here we are using 10. The next two arguments are min and max values for which to create a random vector. Min and Max have defaults of 1 and 10. But we can assign different values for those when calling the function and inputing new values to the arguments. In this case, we use a Min of 100 and a Max of 2000). The arguments you enter take precedent over the default min and max values. The defaults are only used when the arguments are not specified by the user when calling the function.

For this lab, I worked with John, Heather, Mandy, Matt 😊