

Chapter 22: Conditional Execution

Exercise 22.3: The first two methods to solve this exercise were relatively straight forward. I picked a value for "n" and then created commands in R that mirrored the equations in the exercise. The third method was more challenging. I used a for loop and the built-in function "prod" to produce values inside a vector "x" whose product would be the desired answer. Because the vector "x" contained the value 0, I had to make that 0 be counted as a 1 (as the exercise's instructions stated) so the "prod" didn't return the answer 0.

R Code:

Method 1

```
> n = 4
> i = c(2:n)
> C_n = factorial(2 * n) / (factorial(n + 1) * factorial(n))
> C_n
```

Method 2

```
> n = 4
> i = c(2:n)
> C_n1 = choose(2 * n, n) / (n + 1)
> C_n1
```

Method 3

```
> n = 4
> i = c(2:n)
> C_n2 = vector("numeric", length(i))
x = vector("numeric", length(i))
for (i in i) {
  x[i] = (n + i)/i
  C_n2 = x[x!=0]
}
> prod(C_n2)
```

```

> #Question 22.3
> n = 4
> i = c(2:n)
>
> #Method 1
> C_n = factorial(2 * n) / (factorial(n + 1) * factorial(n))
> C_n
[1] 14
>
>
>
> #Method 2
> C_n1 = choose(2 * n, n) / (n + 1)
> C_n1
[1] 14
>
>
> #Method 3
> n = 4
> i = c(2:n)
>
> C_n2 = vector("numeric", length(i))
> x = vector("numeric", length(i))
> for (i in i) {
+   x[i] = (n + i)/i
+   C_n2 = x[x!=0]
+ }
>
> prod(C_n2)
[1] 14
> |

```

values

C_n	14
C_n1	14
C_n2	num [1:3] 3 2.33 2
i	4L
n	4
x	num [1:4] 0 3 2.33 2

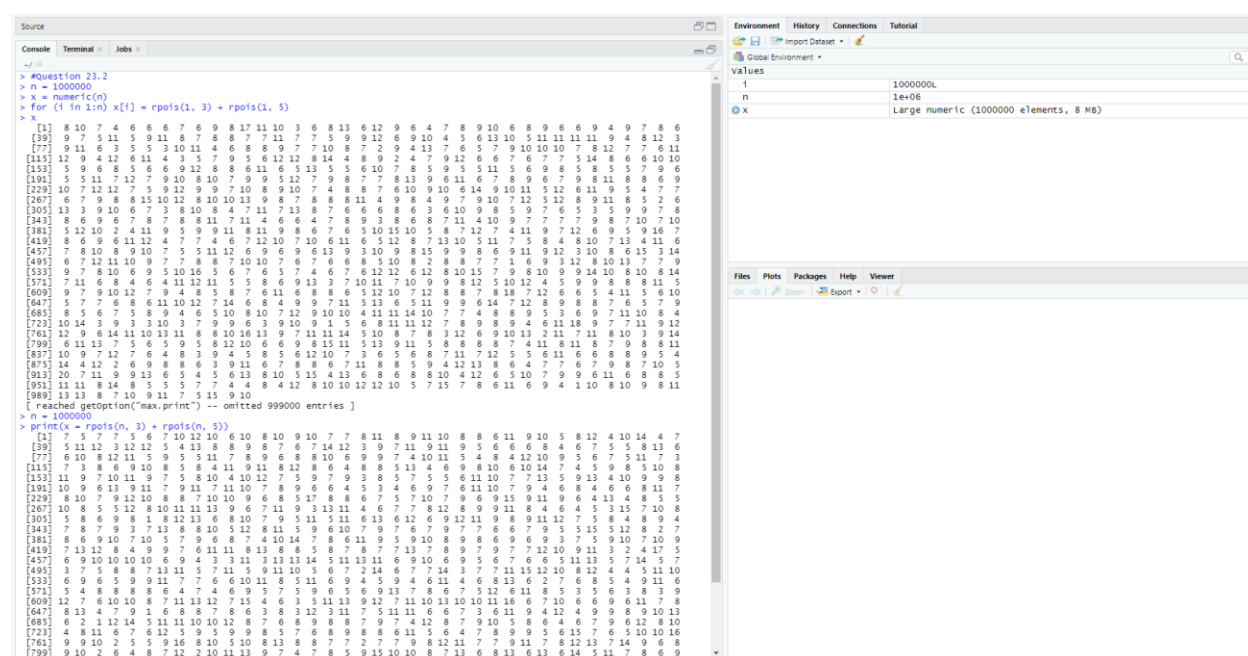
Chapter 23: Iteration

Question 23.4: The "rpois" indicates that this is Poisson distribution. A Poisson distribution models the number of times an event occurs within a specified interval or time. As the code is shown in the book, the user is essentially deriving one value at a time for "n" times. Therefore, by stating the number of times ("n") up front, we cut out extraneous steps/code. Because the directions tell the user to "vectorize this computation" I set my answer to the vector "x." I checked my answer by running the code in Question 23.4. The printed data matched.

R Code:

```
> n = 1000000
> print(x = rpois(n, 3) + rpois(n, 5))
```

```
[ reached getOption("max.print") -- omitted 999900 entries ]
> n = 1000000
> print(x = rpois(n, 3) + rpois(n, 5))
[1] 7 5 7 7 5 6 7 10 12 10 6 10 8 10 9 10 7 7 8 11 8 9 11 10 8 8 6 11 9 10 5 8 12 4 10 14 4 7
[39] 5 11 12 3 12 12 5 4 13 8 8 9 8 7 6 7 14 12 3 9 7 11 9 11 9 5 6 6 6 8 4 6 7 5 5 8 13 6
[77] 6 10 8 12 11 5 9 5 5 11 7 8 9 6 8 8 10 6 9 9 7 4 10 11 5 4 8 4 12 10 9 5 6 7 5 11 7 3
[115] 7 3 8 6 9 10 8 5 8 4 11 9 11 8 12 8 6 4 8 8 5 13 4 6 9 8 10 6 10 14 7 4 5 9 8 5 10 8
```



Question 23.8: The elements of this question were confounding and took me quite some time to figure out. The simple part was making the "checkerboard" function and giving the two arguments "color1" and "color2". After that, I used the "n" type plot because this provides

me a blank canvas on which to assign characteristics. The "xlim" and "ylim" of (1,9) troubled me for a while, as a checkerboard is 8 x 8. However, when I used (1, 8), R didn't return enough squares. I had to use (1, 9) to get an 8 x 8 board. Next, I began my "for" loop. Essentially the remainder of the code is saying: for every integer from 1 through 8 I want the block color to be "color1" if the integer is an odd number; "if else" I want that block color to be "color 2."

R Code:

```
> checkerboard = function(color1, color2){
  plot(0, 0, type = "n", xlim = c(1, 9), ylim = c(1, 9), xlab = "",
  ylab= "", axes = FALSE)

  for (i in 1:8) {

    a = if (i %% 2) c(color1, color2) else c(color2, color1)

    rect(i, 1:9, i+1, 9, col = a)

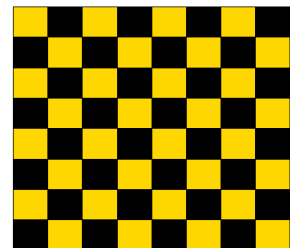
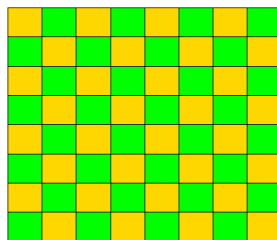
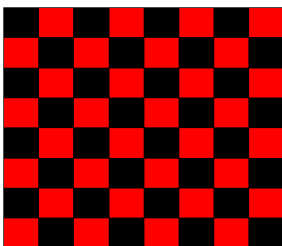
  }}

> checkerboard("red","black")

> checkerboard("green","gold")

> checkerboard("black", "gold")
```

```
> #Question 23.8
> checkerboard = function(color1, color2){
+   plot(0, 0, type = "n", xlim = c(1, 9), ylim = c(1, 9), xlab = "", ylab= "", axes = FALSE)
+   for (i in 1:8) {
+     a = if (i %% 2) c(color1, color2) else c(color2, color1)
+     rect(i, 1:9, i+1, 9, col = a)
+   }
+ }
> checkerboard("red","black")
> checkerboard("green","gold")
> checkerboard("black", "gold")
> |
```



Chapter 24: Recursion

Question 24.2: To solve this problem, I first had to understand what a recursive function is. To wit, I reread Chapter 24 and, per page 189, determined that a recursive function is one that calls on itself until a "terminating base case can be reached." Therefore, I modeled my answer after the online lectures where if the initial "n" is less than or equal to 1, then "n" (1) is returned. In all other cases ("else"), the "n" is operated on in such a way that the sum of (n-1) and (n-2) is returned; this is consistent with characteristics of a Fibonacci number where one value is equal to the sum of the two values preceding it.

R Code:

```
> Fib = function(n) {  
  if(n <= 1){ return(n)}  
  else {  
    return (Fib(n-1) + Fib(n-2))  
  }  
}
```

```
> #Question 24.2  
> Fib = function(n){  
+   if(n <= 1){ return(n)}  
+   else {  
+     return (Fib(n-1) + Fib(n-2))  
+   }  
+ }  
> Fib(10)  
[1] 55  
> Fib(2)  
[1] 1  
> Fib(3)  
[1] 2  
> Fib(1)  
[1] 1  
> Fib(20)  
[1] 6765  
> |
```