Stat 243: Problem Set 1, Due Wednesday Sep-11

August 29, 2016

This covers material in weeks 1 and 2: R data types, vectors, atomic structures, control flow structures, and basic functions.

1. Use an `.Rmd` (Rmarkdown) or an `.Rnw` (R no-web) file for your source code.
2. Your solution should not just be code—you should have text describing how you approached the problem and what the various steps were.
3. In addition to the paper submission, which is the primary thing we will grade, please turn in your knited (compiled) file electronically on bCourses (a file just containing the code with indication of what pieces of code are for what purposes is fine too).
4. All your code should have comments indicating what each function or block of code does, and for any line of code or code constructs that may be hard to understand, a comment indicating what that code does.
5. Use functions as much as possible, in particular for any repeated tasks. We will grade in part based on modularity of your code and your use of functions.

Please check the comments in the syllabus about when to ask for help and about working together.

Questions

1) You can use the colon operator `:` to generate a sequence of values. For instance:

```
1:3
```

```
## [1] 1 2 3
```

```
3:-3
```

```
## [1]  3  2  1  0 -1 -2 -3
```

a. How can you get the help documentation for the colon operator?
b. Why does `1.3:3.2` return the vector `1.3 2.3` and not `1.3 3.2`?
c. Why does `identical()` return `FALSE` when testing if `a` and `b` are the same in the code below?

```
a = c(1, 2, 3, 4, 5)
b = 1:5

identical(a, b)
```

```
## [1] FALSE
```

2) Let $h(x, n) = 1 + x + x^2 + x^3 + \cdots + x^n = \sum_{i=0}^{n} x^i$.

a. Write R code to calculate $h(x, n)$ using a for loop.

b. Write code to achieve the same result in part a) but using a `while` loop.

c. Write code to implement $h(x, n)$ using vector operations (and no loops).

d. The function $h(x, n)$ is the finite sum of a geometric sequence. It has the following explicit formula, for $x \neq 1$,

$$h(x, n) = \frac{1 - x^{n+1}}{1 - x}$$

Write R code to calculate the formula and test it with your code using the following values:

- $h(0.3, 55) = 1.428571$
- $h(6.6, 8) = 4243336$

3) A room contains 100 toggle switches, originally all turned off. These switches can be initialized in R with the following character vector `switches`:

```
num_switches <- 100
switches <- rep("off", num_switches)
```

100 people enter the room in turn. The first person toggles every switch, the second toggles every second switch, the third every third switch, and so on, to the last person who toggles the last switch only. Write R code to find out, at the end of this toggling process, which switches are turned on.

In addition to using the vector `switches`, you can use any control flow structure: if-then-else statements, function `switch()`, `for` loops, `repeat` loops, or `while` loops.

4) In this question we simulate the rolling of a die. To do this we use: `sample(1:6, size = 1)`

a. Suppose that you are playing the gambling game of the Chevalier de Mere. That is, you are betting that you get at least one six in four throws of a die. Write a function `play()` that simulates one round of this game and prints out whether you win or lose. Check that your function can produce a different result each time you run it.

For instance, here are three runs of my `play()` function:

```
play()
```

```
## [1] "you lose"
```

```
play()
```

```
## [1] "you win"
```

```
play()
```

```
## [1] "you lose"
```

b. Now write a more general function `sixes()` that returns `TRUE` if you obtain one six in n rolls of a fair die, and returns `FALSE` otherwise. That is, the argument is the number of rolls n, and the value returned is `TRUE` if you get at least one six and `FALSE` otherwise. Make sure to check that n is a positive integer, and give it a default value of 4.

c. Finally, write code that uses your function `sixes()` from part b), to simulate N plays of the game (each time you bet that you get at least one six in n rolls of a fair die). Your code should then determine the proportion of times you win the bet. This proportion is an estimate of the *probability* of getting at least one six in n rolls of a fair die. Run the code for $n = 4$ and $N = 100$, 1000, and 10000, conducting several runs for each N value. How does the variability of your results depend on N?