Lab 3: For Loops, If Statements, and Monte Carlo Simulation

Lab Goals

The purpose of this lab is to learn about `for` loops, `if` statements, and general "logical operations" in R. Knowing these topics will greatly increase the power of what you can do in R. You will find these topics useful when working with data and when performing computer "experiments" that are called "Monte Carlo simulations." In fact, in this lab we will see our first examples of Monte Carlo simulations.

The `for` loop in R

Sometimes when you are programming, you will want to repeat a similar task many times. For example, suppose you want to calculate $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$ for all values of n up to 50. We know that to get from, say, 9! to 10!, you just need to multiply 9! by 10. Likewise, $n!$ is just $n \times (n-1)!$ So one could do the following:

```
fact <- 1
print(paste0("1! is ", fact))
```

```
## [1] "1! is 1"
```

```
fact <- 2 * fact
print(paste0("2! is ", fact))
```

```
## [1] "2! is 2"
```

```
fact <- 3 * fact
print(paste0("3! is ", fact))
```

```
## [1] "3! is 6"
```

```
fact <- 4 * fact
print(paste0("4! is ", fact))
```

```
## [1] "4! is 24"
```

```
# and so on!!
```

In the above `paste0` combines its arguments into a string. Then `print` outputs this string to the console (or adds it to the Rmarkdown output).

But this is very tedius! Clearly there's a pattern that we want to repeat. Fortunately, computers are good at repeating tedius tasks. The `for` loop is a way of getting R to repeat something for you over and over.

```
fact <- 1
for (i in 1:50) {
  fact <- i * fact
  print(paste0(i, "! is ", fact))
}
```

```
## [1] "1! is 1"
## [1] "2! is 2"
## [1] "3! is 6"
## [1] "4! is 24"
## [1] "5! is 120"
## [1] "6! is 720"
## [1] "7! is 5040"
## [1] "8! is 40320"
## [1] "9! is 362880"
## [1] "10! is 3628800"
## [1] "11! is 39916800"
## [1] "12! is 479001600"
## [1] "13! is 6227020800"
## [1] "14! is 87178291200"
## [1] "15! is 1307674368000"
## [1] "16! is 20922789888000"
## [1] "17! is 355687428096000"
## [1] "18! is 6402373705728000"
## [1] "19! is 121645100408832000"
## [1] "20! is 2432902008176640000"
## [1] "21! is 51090942171709440000"
## [1] "22! is 1.12400072777761e+21"
## [1] "23! is 2.5852016738885e+22"
## [1] "24! is 6.20448401733239e+23"
## [1] "25! is 1.5511210043331e+25"
## [1] "26! is 4.03291461126606e+26"
## [1] "27! is 1.08888694504184e+28"
## [1] "28! is 3.04888344611714e+29"
## [1] "29! is 8.8417619937397e+30"
## [1] "30! is 2.65252859812191e+32"
## [1] "31! is 8.22283865417792e+33"
## [1] "32! is 2.63130836933694e+35"
## [1] "33! is 8.68331761881189e+36"
## [1] "34! is 2.95232799039604e+38"
## [1] "35! is 1.03331479663861e+40"
## [1] "36! is 3.71993326789901e+41"
## [1] "37! is 1.37637530912263e+43"
## [1] "38! is 5.23022617466601e+44"
## [1] "39! is 2.03978820811974e+46"
## [1] "40! is 8.15915283247898e+47"
## [1] "41! is 3.34525266131638e+49"
## [1] "42! is 1.40500611775288e+51"
## [1] "43! is 6.04152630633738e+52"
## [1] "44! is 2.65827157478845e+54"
## [1] "45! is 1.1962222086548e+56"
## [1] "46! is 5.50262215981209e+57"
## [1] "47! is 2.58623241511168e+59"
## [1] "48! is 1.24139155925361e+61"
## [1] "49! is 6.08281864034268e+62"
## [1] "50! is 3.04140932017134e+64"
```

In words, the code above starts with `fact` assigned as 1. It then enters the `for` loop. The line `for (i in 1:50)` says that we will repeat whatever is between `{` and `}`. In particular, we will first do it with `i` equal to 1, then with `i` equal to 2, etc, and (finally) with `i` equal to 50.

So the code above is equivalent to

```
fact <- 1
i <- 1
fact <- i * fact
print(paste0(i, "! is ", fact))
```

```
## [1] "1! is 1"
```

```
i <- 2
fact <- i * fact
print(paste0(i, "! is ", fact))
```

```
## [1] "2! is 2"
```

```
i <- 3
fact <- i * fact
print(paste0(i, "! is ", fact))
```

```
## [1] "3! is 6"
```

```
# etc... until we get to i being 50
```

Logical statements and the `if` statement

Sometimes we want R to do something only in certain cases. An `if` statement is a way of specifying that a block of code should only be run if a given logical statement is true. A logical statement is something that returns either `TRUE` or `FALSE`. Such statements involve logical operators. Here is a table of logical operators in R.

Operator	Description
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	exactly equal to
`!=`	not equal to
`!x`	not x
`x | y`	x OR y
`x & y`	x AND y

Here are some examples of how R evaluates these operators.

```
x <- 3
y <- 6
z <- 6

x < y
```

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

```
y < x
```

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

```
x == y # notice the "double equals" is used for logical statements
```

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

```
!(x == y)
```

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

```
y == z
```

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

```
y <= z
```

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

```
y >= z
```

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

```
(x < y) & (y < (x + 1))
```

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

```
(x < y) | (y < (x + 1))
```

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

Question: Why do you think `==` is used above (for example, `x == y`)? Check in the console what would happen if we used the single equals sign.

The single `=` is used for assignment. So `x = y` would assign `x` to have the value of `y`, which in this case would change `x` to have value 6.

The syntax of an `if` statement is as follows

`if (logical statement) {commands}`.

Suppose we want to print "Yes" if the third element of this vector is less than 10.

```
a <- c(2, 5, 6, 88, 3, 20)
```

We can do so with an `if` statement as follows:

```
if (a[3] < 10) {
  print("Yes")
}
```

```
## [1] "Yes"
```

Sometimes we will want to run some alternate code when the condition does not hold. For this, we can add `else` to the `if` statement as follows:

```
if (a[3] < 10) {
  print("Yes")
} else {
  print("No")
}
```

```
## [1] "Yes"
```

We can try this for the fourth element of `a` to see what happens when the condition does not hold.

```
if (a[4] < 10) {
  print("Yes")
} else {
  print("No")
}
```

```
## [1] "No"
```

Problem 1

Suppose we want to print "Yes" or "No" for each element in the vector `a`. Write a `for` loop that does this.

```
for (i in 1:length(a)) {
  if (a[i] < 10) {
    print("Yes")
  } else {
    print("No")
  }
}
```

```
## [1] "Yes"
## [1] "Yes"
## [1] "Yes"
## [1] "No"
## [1] "Yes"
## [1] "No"
```

Yahtzee

There's a game played with five dice called Yahtzee. There are a lot of rules involved, but for the sake of this lab all you need to know is that you roll the five dice and you "get a Yahtzee" if all five dice have the same value.

R has many functions that can be used to "simulate" random occurrences. One of these functions is called `sample`. The following code simulates rolling five six-sided dice.

```
sample(1:6, 5, replace = TRUE)
```

```
## [1] 4 2 1 6 2
```

Just like when rolling dice, each time we do it, we get a different answer:

```
sample(1:6, 5, replace = TRUE)
```

```
## [1] 1 2 6 1 5
```

```
sample(1:6, 5, replace = TRUE)
```

```
## [1] 5 1 3 2 5
```

Problem 2

Write a code chunk that

a) simulates the rolling of five dice and assigns the result to a variable named `dice_rolls`.

b) Then write a logical expression that is `TRUE` if `dice_rolls` is a Yahtzee (that is, if all five dice are equal in value to each other) and is `FALSE` otherwise.

```
dice_rolls <- sample(1:6, 5, replace = TRUE)
(dice_rolls[1] == dice_rolls[2]) & (dice_rolls[1] == dice_rolls[3]) & (dice_rolls[1] == dice_rolls[4]) &
```

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

Note: There are many ways to do part b. One approach is to check that each die equals the first die (and use `&`). Another approach is to check if the standard deviation is 0. Try writing out both approaches for practice.

```
sd(dice_rolls) == 0
```

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

Problem 3

We are curious to know approximately what fraction of the time one gets all five dice equaling each other. To answer this question, we want to repeat Problem 2 a very large number of times (say one hundred thousand times) and count how many times we get Yahtzees. We can use a `for` loop to do this.

```
set.seed(123) # adding this so answers don't change
number_of_simulations <- 100000
number_of_yahtzees <- 0
for (i in 1:number_of_simulations) {
  dice_rolls <- sample(1:6, 5, replace = TRUE)
```

```
  if (sd(dice_rolls) == 0) { # checks if they are all equal to each other
    number_of_yahtzees <- number_of_yahtzees + 1
  }
}
number_of_yahtzees
```

```
## [1] 87
```

```
number_of_yahtzees / number_of_simulations # proportion of Yahtzees
```

```
## [1] 0.00087
```

Approximately what proportion of the time does one expect to get a Yahtzee?

One expects to get a Yahtzee about 0.09% of the time.

Note: We have just performed a **Monte Carlo simulation**! This refers to using a (pseudo)random number generator to observe the fraction of the time an event occurs. This is a useful technique that will allow us to answer many questions related to probability and statistics.

Problem 4

Suppose you roll only two dice. Write a code chunk to see approximately what fraction of the time the two rolled dice are equal.

```
number_of_simulations <- 100000
number_of_times_equal <- 0
for (i in 1:number_of_simulations) {
  dice_rolls <- sample(1:6, 2, replace = TRUE)
  if (dice_rolls[1] == dice_rolls[2]) {
    number_of_times_equal <- number_of_times_equal + 1
  }
}
number_of_times_equal
```

```
## [1] 16776
```

```
number_of_times_equal / number_of_simulations # proportion of times they are equal
```

```
## [1] 0.16776
```

Problem 5

Suppose you roll two dice. Write a code chunk to see approximately what fraction of the time are neither of the two dice are equal to 6.

```
number_of_simulations <- 100000
number_of_times_both_not_six <- 0
for (i in 1:number_of_simulations) {
  dice_rolls <- sample(1:6, 2, replace = TRUE)
  if ((dice_rolls[1] != 6) & (dice_rolls[2] != 6)) {
```

```
    number_of_times_both_not_six <- number_of_times_both_not_six + 1
  }
}
number_of_times_both_not_six
```

```
## [1] 69618
```

```
number_of_times_both_not_six / number_of_simulations
```

```
## [1] 0.69618
```