Lab 1: Data Types and Control Statements

Jon Ashbrock

January 18, 2018

1 Introduction

By the end of this lab students will:

1. Be able to create and interpret list and array objects
2. Be able to generate random numbers in **R**
3. Be able to understand and use the conditional (IF) statement

2 Lists and Arrays in R

In almost all of our work through the semester we will want to store our data points as one variable. Therefore we need a way for variables to store multiple numbers. The method we will use to do this is the list. We have already seen one way of making a list in the for-loop. The syntax "1 : 10" creates a vector of the integers starting at 1 and ending at 10. Consider the following lines of code: The above code will turn the variable vec into a list

```
vec <- 1:100
print(vec[10])
```

containing the integers starting at 1 and ending at 100. Then, the second line will print the tenth element in the vector, which happens to be the number 10. However if I type the code at the beginning of the next page into the console, then the output will be the number 9. The syntax "vec[10]" tells the computer to find and return whatever the tenth element in the list is.

```
vec <- 0:100
print(vec[10])
```

Another nice way to create a list is to use the concatenate command. If I want to create a list containing the number $1, 2, 4, 8, 16$ then I can type:

```
vec <- c(1,2,4,8,16)
print(vec[3])
```

These two lines together would print the value "4" (the third element in the list).

Sometimes we also may want to generate matrices or "n-dimensional matrices" (which are called arrays) to store our data. The following code uses the parameter "dim" to define what your array should look like. Notice that

```
mat <- array(0, dim = c(a,b,c))
```

the above array will be 3 dimensional and dimension one will have size "a", the second dimension has size "b" and dimension three has size "c".

A last, simple function we can use to generate matrices is the "matrix" function. To define a matrix we need to specify the values for "nrow" and "ncol" which correspond to the number of rows and columns respectively. If I want to create a matrix of $0's$ that has 4 rows and 5 columns, I could type:

```
My_Matrix <- matrix(0, nrow=4, ncol=5)
```

Discussion. *Write a program to create a (5x5) matrix where the (i, j) entry contains the value $i \cdot j$.*

3 Generating Random Numbers

Computers are not able to generate a random numbers. However, computers can generate a sequence of numbers which is "as good as possible" when it comes to being random. If this is the case, then, why do we say a computer can't generate random numbers?

Computers are only able to do one thing: calculations. They can add, multiply, subtract, divide, and look up numbers stored in a table. When a computer generates random numbers it uses an algorithm (look up the Mersenne-Twister) to produce numbers which look random. However, since the computer uses an algorithm to compute the random numbers it is possible to predict them. That is, if you know what value the computer started at in the algorithm then you could with 100% accuracy predict the sequence of random numbers. This is why the numbers are called *pseudo*-random numbers.

However, for all intents and purposes the numbers that a computer generates can be thought of as truly random numbers. There are many methods to doing this and we will introduce them as necessary. The first is generating lists of uniformly distributed random numbers. We will use the *runif* command to do this. *runif* accepts three arguments as input. The first is how many numbers you want to generate and the second and third are the two bounds on the interval you want random numbers from. So, if I want to store 10 numbers drawn uniformly at random from the interval $[0, 1]$ in a variable, I can do this with the command

```
Random_Numbers <- runif(10,0,1)
```

One more way to generate random numbers is using the *sample* function. *sample* is a function which accepts three arguments as input. The first is a list which the function will sample from, the second specified how many samples to take, the third tells you whether or not to replace elements to the list after they have been selected. If I want to generate 100 random integers between 1 and 10 inclusive (with replacement), the following code will do just that:

```
My_Sample <- sample(1:10, 100, replace=T)
```

4 The IF-statement

It is useful to be able to check whether or not a condition is true or not before executing a line of code. To illustrate an example, let us consider the following example:

Discussion. *Suppose we want to estimate the probability that a random integer chosen in the set* $\{1, \ldots, 10\}$ *is at least as large as* 5. *One method is to generate a large number of random numbers from this set and count how many are larger than* 5. *Write the Pseudo-code (step-by-step instructions) to do this and get our answer.*

To do this we need to determine whether or not a number is larger than 5. **R** has many built in logical operators that will return a true-or-false output. These are $<, <=, >, >=, ==, != $ and they will perform the logical check that you expect them to. An IF-statement is provided a logical statement as input and if that statement is true, the statements inside the IF-statement are executed, otherwise they are skipped. To do the check whether or not a number is at least 5, we can write the following code:

```
if (Number >= 5){
# Statements to execute
}
```

Discussion. *Now let us write the* ***R*** *code for the problem explained at the beginning of this section.*

If we wanted to check for multiple values we can connect two logical statements with the && (and) or || (or) symbols. So, if instead we wanted to count the numbers greater than or equal to 5 but smaller than 8 we could type

```
if (Number >= 5 && Number <8){
```