Stat 110: Section 1

Adapted from Joe Blitzstein and Jessica Hwang's Introduction to Probability and Kenneth Baclawski's Introduction to Probability with R

Justin Zhu

August 25, 2018

Getting familiar with R

You can follow along by typing the scripts into a console or an R file in RStudio IDE. This R Markdown notebook will contain the R script on one line and the subsequent output in the next line. You can press "Run" to run the selected lines while interacting with this RMarkdown file.

For example,

```
1+1
```

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

The R code is "1+1" and the output looks something like "##[1] 2".

Some more scripts to play around with:

```
2 * pi * 5 + 100
```

```
## [1] 131.4159
```

```
100 * 299792458 ^ 2
```

```
## [1] 8.987552e+18
```

R follows conventional use of mathematical operators and order of operations.

R is built around vectors.

A simple vector is the sequence 1, 2, . . ., n. If you want n to be 100, then we type this command:

```
1:100
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
```

```
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

We also use the `c` command to combine and concactenate values into a vector.

```
c(1,4,9,16,25,36)
```

```
## [1]  1  4  9 16 25 36
```

Here, we stored those values into v.

```
v <- c(1,4,9,16,25,36)
```

We can access elements inside v using the following:

```
v[2]
```

```
## [1] 4
```

We can also find several values within v an form a new vector

```
v[c(1,3,5)]
```

```
## [1]  1  9 25
```

R does *not* use zero-index.

Variables and Functions

We have already seen how "<-" is used in R to store values into a variable. You can also use "->" in the other direction. The arrow, however must point to the variable is assigned the value. For example, in this example, triplepie and pietriple are both being assigned the same values.

```
triplepie <- 3*pi

3*pi -> pietriple
```

We can use boolean operators to verify whether certain variables contain the same values

```
triplepie == pietriple
```

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

The boolean operators in R also follow conventional programming language practice. You can find a list of them here

To create a function we can use the "function" keyword and specify parameters. We can write a function that calculates the probability of at least one birthday match in a group of n people.

```r
birthday <- function(n) {
  days_in_year <- 365
  ans <- 1-prod((days_in_year-n+1):days_in_year)/days_in_year^n
  return(ans)
}
```

Here, `prod` returns the product of all the values passed in as arguments.

You can always use help() to find more information about R's built-in functions.

```r
help(prod)
```

We can now finally celebrate some birthdays by typing our function name with the specified parameters!

```r
birthday(1)
```

```
## [1] 0
```

```r
birthday(23)
```

```
## [1] 0.5072972
```

How can we plot the relationship between number of people and the probability of getting a matching birthday? How does changing the number of people in our birthday problem affect the probability?

```r
num_of_people <- 1:50
matching_birthdays_probability <- lapply(num_of_people, birthday)
plot(num_of_people,matching_birthdays_probability)
```