# 1

# THE NATURE OF RANDOMNESS

Probability theory, and its practical applications all center around randomness. But what is randomness?

Suppose you have die with ten faces, each face of which is a unique number, stretching from 1 to 10. Let us throw two such dice, at the same time. What will be the outcome?

After installing the Julia programming language, we run this simulation:

```
rand(1:10, 2)
```

In Julia, rand( ) is a built-in function (tool) that takes inputs and produces random results. The 1:10 is an inclusive range from 1 to 10 obviously, and 2 stands for how many things we are experimenting with.

Here is the output:

```
2-element Array{Int64,1}:
 6
 8
```

While Julia gave us 6 and 8, actually the output does not consist of two numbers, but a single thing called an array, and whose members are called elements. Iin other words, the output is a 2-element array.

An array, in Julia, can be considered as a list of things, such as numbers. So, Julia says the output is an array [ ] (anything enclosed in square brackets is called an array) , filled with two integers.

As for Int64, Julia says it is an array of the type that holds high-precision integers, so that there are no calculation errors.

What would happen if you threw it, say, 10 times?

```
[rand(1:10, 2) for i in 1:10]
```

Note that the two after the comma stands for throwing two dice once, or one die twice. From a probabilistic viewpoint, they are both the same.

Here I used what is called a comprehension. In the first part involving *rand()*, which you are familiar with, we throw our two dice. Then using *for i* we are repeating this throw one to ten times.

Can I use anything else instead of *i*?

Yes. We call this an iterator dummy. In simpler terms, it goes over each **thing** at least once, whereas in this case, the **thing** is decided by the *rand(1:10, 2)* function in front of it.

Our output is:

```
10-element Array{Array{Int64,1},1}:
 [3, 10]
 [9, 3]
 [9, 8]
 [10, 8]
 [6, 4]
 [1, 2]
 [10, 3]
 [2, 2]
 [7, 10]
 [8, 5]
```

Randomness is real, and throwing the dice a few times, we cannot predict what is going to be next. So what is exactly the point of understanding probability, then? The point is that randomness is not as random as it appears.

Let us throw a six-sided die, ten times. It will be impossible for you to predict the the eleventh throw. But what happens if we throw the same die 10000 times? How often would a six occur in 10000 throws?

```
occurs = 1

for k in 1:10
        println(sum(
               [occurs for i in 1:10000
                     if rand(1:6)==5]))
        end
```

The code is slightly more difficult than the previous comprehension. First, we create a variable called occurs, and assign to it the value of 1. This will help us with counting, when we count how many times the six-sided die lands on five.

Next, we use an iterator dummy *k*, to give us ten iterations of a certain activity. That activity is defined as printing on a new line, the sum of all occurrences of 5, when we repeat the die throw 10000 times.

In the jargon, we have run a standard *for* loop on a nested function conditional comprehension. We nested the *sum()* within *println()*, and our comprehension has a condition (*if* something, do this) at its end.

Finally, the standard *for* loop must become closed; in Julia, we write end to close off loops and functions. You have also noticed that the *for* loop inside the comprehension does not require and *end*.

Also note, that I have written the code in this style to make it easy to decipher. It is not a standard or idiomatic Julia style, but more on this in the future.

Let us now inspect the output:

```
1628
1676
1706
1649
1672
1635
1599
1653
1642
1608
```

What is worth noticing here? We know, from the classical definition, that the probability of landing a five, when we throw a six-sided die is 1/6, since only one of the six possibilities is a five.

Whaat the 10000 throws show, however, is remarkable: dividing the output we have above, over the total number of throws, we get an interesting result.

Let us place the output in a new array, aptly called output:

```


output = [1628
          1676
          1706
          1649
          1672
          1635
          1599
          1653
          1642
          1608]
```

In Julia, element-wise operations can be achieved directly, so no loops are needed here. We simply divide each element on 10000, like so:

```
output ./ 10000
```

The dot in front of the division symbol will ensure that each element of the output array is divided over 10000.

The result is:

```
  0.1628
  0.1676
  0.1706
  0.1649
  0.1672
  0.1635
  0.1599
  0.1653
  0.1642
  0.1608
```

Almost the same as the classical probability!

The main point is that empirical probability, or what actually happens in the real world, mimics classical, pre-defined probabilities when the number of experiments increases. In other words, over the long run, empirical probabilities converge on the classical.