A Concrete Introduction to Probability (using Julia)

Preface

This lesson was originally ***authored by Peter Norvig for the Python programming language****. While this notebook aims to recreate that lesson as faithfully as possible, in content and in spirit – a lot of the text is almost verbatim one-to-one with the original as written by Dr. Norvig – certain liberties need to be taken when translating code from one language to another. The two languages are syntactically similar enough, yet certain concepts don't translate neatly from one to the other. In any case, I hope you enjoy reading (and interacting!) with this notebook for I find probability, Pluto, and Julia all quite beautiful.*

In 1814, Pierre-Simon Laplace **wrote**:

> Probability theory is nothing but common sense reduced to calculation. ... [Probability] is thus simply a fraction whose numerator is the number of favorable cases and whose denominator is the number of all the cases possible ... when nothing leads us to expect that any one of these cases should occur more than any other.

Laplace nailed it. To untangle a probability problem, all you have to do is define exactly what the cases are, and careful count the favorable and total cases. Let's be clear on our vocabulary words:

- **Trial**: A single occurrence with an outcome that is uncertain until we observe it. *For example, rolling a single die.*

- **Outcome**: A possible result of a trial; one particular state of the world. What Laplace calls a **case**. *For example: 4.*
- **Sample Space**: The set of all possible outcomes for the trial. For example, {1, 2, 3, 4, 5, 6}.
- **Event**: A subset of outcomes that together have some property we are interested in. *For example, the event "even die roll" is the set of outcomes {2, 4, 6}.*
- **Probability**: As Laplace said, the probability of an event with respect to a sample space is the "number of favorable cases" (outcomes from the sample space that are in the event) divided by the "number of all the cases" in the sample space (assuming "nothing leads us to expect that any one of these cases should occur more than any other"). Since this is a proper fraction, probability will always be a number between 0 (representing an impossible event) and 1 (representing a certain event). *For example, the probability of an even die roll is 3/6 = 1/2.*

This notebook will explore these concepts in a concrete way using Julia code. The code is meant to be succint and explicit, and fast enough to handle sample spaces with millions of outcomes. If you need to handle trillions, you'll want a more efficient implementation. Peter Norvig also has **another notebook** that covers paradoxes in Probability Theory.

P is for Probability

The code below implements Laplace's quote directly:

> Probability is thus simply a **fraction** whose numerator is the **number of favorable cases** and whose denominator is the **number of all the cases possible**.

```julia
"""
The probability of an event, given a sample space.
"""
function P(event::Set, space::Set)

    # favorable = outcomes that are in the event and in the sample space
    favorable = intersect(event, space)

    # the number of cases is the length, or size, of a set
    num = length(favorable)
    den = length(space)

    # this returns the result as a Rational
    return num // den
end;
```

Note

Read more about Rational numbers in Julia ***here***

Warm-up Problem: Die Roll

What's the probability of rolling an even number with a single six-sided fair die? Mathematicians traditionally use a single capital letter to denote a sample space; we'll use D for the die:

```
D =  Set([5,  4,  6,  2,  3,  1])
```

```julia
D = Set([1, 2, 3, 4, 5, 6])
```

```
even =  Set([4,  6,  2])
```

```julia
even = Set([2, 4, 6])
```

```
1//2
```

```julia
P(even, D)
```

Good to confirm what we already knew. We can explore some other events:

```
prime =  Set([5,  13,  7,  2,  11,  3])
```

```julia
prime = Set([2, 3, 5, 7, 11, 13])
```

```
odd =  Set([5,  13,  7,  11,  9,  3,  1])
```

```julia
odd = Set([1, 3, 5, 7, 9, 11, 13])
```

```
1//2
```

```julia
P(odd, D)
```

```
5//6
```

```julia
P(union(even, prime), D)
```

```
1//3
```

```julia
P(intersect(odd, prime), D)
```

Note

Read more about Sets in Julia ***here****, and in general* ***here****. And then there are* ***Julia Sets****, but that's a different thing....*

Card Problems

Consider dealing a hand of five playing cards. An individual card has a rank and suit, like **J♥** for the Jack of Hearts, and a deck has 52 cards:

```
begin
    suits = "♥♠♦♣"
    ranks = "AKQJT98765432"
end;
```

```
deck = 13×4 Matrix{Tuple{Char, Char}}:
        ('A', '♥')  ('A', '♠')  ('A', '♦')  ('A', '♣')
        ('K', '♥')  ('K', '♠')  ('K', '♦')  ('K', '♣')
        ('Q', '♥')  ('Q', '♠')  ('Q', '♦')  ('Q', '♣')
        ('J', '♥')  ('J', '♠')  ('J', '♦')  ('J', '♣')
        ('T', '♥')  ('T', '♠')  ('T', '♦')  ('T', '♣')
        ('9', '♥')  ('9', '♠')  ('9', '♦')  ('9', '♣')
        ('8', '♥')  ('8', '♠')  ('8', '♦')  ('8', '♣')
        ('7', '♥')  ('7', '♠')  ('7', '♦')  ('7', '♣')
        ('6', '♥')  ('6', '♠')  ('6', '♦')  ('6', '♣')
        ('5', '♥')  ('5', '♠')  ('5', '♦')  ('5', '♣')
        ('4', '♥')  ('4', '♠')  ('4', '♦')  ('4', '♣')
        ('3', '♥')  ('3', '♠')  ('3', '♦')  ('3', '♣')
        ('2', '♥')  ('2', '♠')  ('2', '♦')  ('2', '♣')
```

```
deck = [(r, s) for r in ranks, s in suits]
```

Note

Experiment a bit in the cell above – see what happens if you change the comma to a `for` in the array comprehension, or see what happens when you execute `deck[:]`, or wrap it in `Set()`, etc.

```
52
```

```
length(deck)
```

Now we want to define Hands as the sample space of all 5-card combinations from deck. The function i `combinations` does most of the work; we than concatenate each combination into a space-separated string:

```
using Combinatorics
```

```
function combos(items, n)
    return Set(combinations(items, n))
end;
```

```
Hands =
  Set([[('A', '♥'), ('Q', '♥'), ('2', '♥'), ('4', '♠'), ('5', '♦')], [('5', '♥'), ('
```

```
Hands = combos(deck, 5)
```

```
2598960
```

```
# number of possible hands
length(Hands)
```

```
Vector{Tuple{Char, Char}}[
                      1:    [('5', '♥'), ('3', '♠'), ('7', '♦'), ('6', '♦'), ('4
                      2:    [('8', '♥'), ('6', '♥'), ('J', '♠'), ('3', '♠'), ('8
                      3:    [('4', '♠'), ('2', '♠'), ('A', '♦'), ('6', '♦'), ('8
                      4:    [('T', '♥'), ('A', '♣'), ('7', '♣'), ('5', '♣'), ('4
                      5:    [('A', '♥'), ('9', '♥'), ('3', '♦'), ('6', '♣'), ('2
                      6:    [('K', '♥'), ('7', '♥'), ('2', '♥'), ('T', '♠'), ('4
                      7:    [('8', '♦'), ('6', '♦'), ('5', '♦'), ('4', '♦'), ('Q
                     ]
```

```
# random sample of hands
rand(Hands, 7)
```

```
[('T', '♥'), ('T', '♠'), ('J', '♥'), ('8', '♣'), ('T', '♠'), ('6', '♦'), ('7', '♦
```

```
# random sample of cards from the deck
rand(deck, 7)
```

Now we can answer questions like the probability of being dealt a flush (5 cards of the same suit):

```
flush =
 Set([[('Q', '♥'), ('7', '♥'), ('6', '♥'), ('5', '♥'), ('2', '♥')], [('K', '♣'), ('
```

```
flush = collect(Hands)[length.(unique.([getfield.(collect(h), 2) for h in Hands])) .==
1] |> Set
```

```
33//16660
```

```
P(flush, Hands)
```

```
Enter cell code...
```

Table of Contents