Contents

CHAPTER 1

Introduction to Discrete Probability

First, let's take care of the word *discrete* vs *discreet*. They sound exactly the same, but "discrete" means "individually separate and distinct" and "discreet" means "careful about what other people know". So you might say, "You can think of light as a continuous wave or as a blast of discrete particles." And you might say, "Please go get the box of doughnuts from the kitchen. Oh, and there are a lot of hungry people in the house, so be discreet."

When we are talking about probabilities, some problems deal with discrete quantities like "What is the probability that I will throw these three dice and the numbers that roll face up sum to 9?". There are also problems that deal with continuous properties like "What is the probability that the next bird to fly over my house will weigh between 97.2 and 98.1 grams ?" In this module, we are going to focus on the probability problems that deal with discrete quantities.

Watch Khan Academy's Introduction to Probability at https://youtu.be/uzkc-qNVoOk.

Let's say that I have a cloth sack filled with 100 marbles; 99 are red and 1 is white. If

I ask you to reach in without looking and pull out one marble, you will probably pull out a red one. We say that "There is a 1 in 100 chance that you would pull out a white marble." Or we can use percentages and say "There is a 1% chance that you will pull out a white marble." Or we can use decimals and say "There is a 0.01 probability that you will pull out a white marble." In probability, we often talk about the probability of certain events. "Pulling out a white marble" is an event, and we can give it a symbol like W. Then, in equations we use p to mean "the probability of". Thus, we can say "There is a 0.01 probability that you will pull out a white marble" which becomes the equation

$$p(W) = 0.01$$

1.1 The Probability of All Possibilities is 1.0

We know that you are either going to pull out a red marble or a white marble, so the probability of a white marble being pulled and the probability of a red marble being pulled must add up to 100%. Therefore, the odds of pulling out a red marble must be 99% or 0.99. If we let the event "Pull out a red marble" be given by the symbol R, we can say:

$$p(R) = 1.0 - P(W) = 1.0 - 0.01 = 0.99$$

Now, let's say that I make you take a marble from the bag and then toss a coin. What is the probability that you will pull a white marble and then get heads on the coin? It is the product of the two probabilities: $0.01 \times 0.5 = 0.005$, so one-half of a one percent chance. Do the probabilities still sum to 1?

- White and Heads = $0.01 \times 0.5 = 0.005$
- White and Tails = $0.01 \times 0.5 = 0.005$
- Red and Heads = $0.99 \times 0.5 = 0.495$
- Red and Tails = $0.99 \times 0.5 = 0.495$

Yes, the probabilitites of all the possibilities still add to 1.

1.2 Independence

In the last section, I told you that the probability of two events ("Pulling a red marble from the bag" and "Getting tails in a coin toss") is the product of the probability of each event: $0.99 \times 0.5 = 0.495$.

This is true if the two events are *independent*, that is the outcome of one doesn't change the probability of the other. The example I gave is independent: It doesn't matter what ball you pull from the bag, the outcome of the coin toss will always be 50-50.

What are two events that are not independent? The probability that a person is a professional basketball player and the probability that someone wears a shoe that is size 13 or larger is *not* independent. After all, height is an advantage in basketball and most tall people also have large feet. So if you know someone is a basketball player, they likely wear large shoes.

Exercise 1 Rolling Dice

If I give you three dice to roll, what is the probability that you will roll a 5 on all three dice?

Working Space

Answer on Page 23

Exercise 2 Flipping Coins

If I give you five coins to flip, what is the probability that at least one coin will come up heads?

Working Space

Answer on Page 23

1.3 Why 7 is the most likely sum of two dice

If you roll two dice, the sum will be 2 or 12 or any number in between. It is very tempting to assume that the likelihood of any of those numbers is the same. In fact, the probability of a 2 is $\frac{1}{36} \approx 3\%$ and the probability of a 7 is $\frac{1}{6} \approx 17\%$. A 7 is six times more likely than a 12! Why?

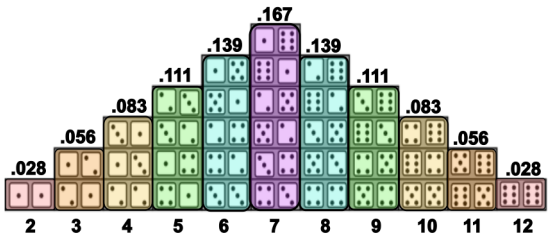
When you roll the first die, there are six possibilities with equal probability. When you roll the second die, there are six possibilities with equal probability. so there are a total of 36 possible events with equal probabilities: 1 then 1, 1 then 2, 2 then 1, 1 then 3, 3 then

1, etc. Only one of these (1 then 1) adds to 2. But six of these sum to 7: 1 then 6, 6 then 1, 2 then 5, 5 then 2, 3 then 4, 4 then 3. So a 7 is six times more likely than a 2.

Here is the complete table:

Sum							Count	Probability
2	1,1						1	1/36
3	1,2	2,1					2	1/18
4	1,3	2,2	3,1				3	1/12
5	1,4	2,3	3,2	4,1			4	1/9
6	1,5	2,4	3,3	4,2	5,1		5	5/36
7	1,6	2,5	3,4	4,3	5,2	6,1	6	1/6
8		2,6	3,5	4,4	5,3	6,2	5	5/36
9			3,6	4,5	5,4	6,3	4	1/9
10				4,6	5,5	6,4	3	1/22
11					5,6	6,5	2	1/18
12						6,6	1	1/36

When I bumped into this, I was skeptical. I decided to test it, so I rolled a pair of dice hundreds of times and made a histogram. It was a tedious and time-consuming task – just the sort of thing that we make computers do for us.



1.4 Random Numbers and Python

You are going to write a simulation of rolling dice in Python. To do this, you will need to generate a random sequence of numbers. The numbers will need to be in the range 1 to 6, and they will need to appear in the sequence with the same frequency. We say the sequence will follow *the uniform distribution*. That is, the probability is uniformly distributed among the 6 possibilities.

Start python and try a few of the different ways to generate random numbers:

```
> python3
>>> import random
>>> random.random()  # Generates a random floating point number between 0 and 1
0.6840892758539989
>>> randrange(5)      # Generates an integer in the range 0 - 4
2
```

```
>>> x = ['Rock', 'Paper', 'Scissors']
>>> random.choice(x)   # Pick a random entry from the sequence
'Paper'
>>> x
['Rock', 'Paper', 'Scissors']
>>> random.shuffle(x)   # Shuffle the order of the sequence
>>> x
['Scissors', 'Paper', 'Rock']
>>> a = list(range(30))
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
  16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 29]
>>> random.sample(a, 10) # Return 10 randomly chosen items from the sequence
[8, 7, 20, 9, 25, 13, 23, 11, 14, 16]
```

Clearly, Python has a lot of ways to do things that look random. I should be honest with you at this point: they aren't really random. The computer that you are using can't generate random data. Instead, it uses tricks to create data that looks random; we call this *pseudorandom* data. Good pseudorandom algorithms are very important for cryptography and data security.

What if you want real random data? Some companies that are using the decay of radioactive materials to generate real random data. You can pay to download it. For our purposes, Python's pseudorandom numbers are quite sufficient.

If we generate two random numbers in the range 1 through 6 and add them together, we will have simulated rolling a pair of dice. Like this:

```
>>> a = random.randrange(6) + 1
>>> b = random.randrange(6) + 1
>>> a + b
8
```

First, let's write a program that just rolls the dice 100 times and shows the result. Make a file `dice.py`:

```
import random

roll_count = 100

for i in range(roll_count):
    a = random.randrange(6) + 1
    b = random.randrange(6) + 1
```

```
    roll = a + b
    print(f"Toss {i}: {a} + {b} = {roll}")
```

When you run it, you should see something like:

```
> python3 dice.py
Toss 0: 6 + 6 = 12
Toss 1: 4 + 4 = 8
Toss 2: 4 + 2 = 6
Toss 3: 4 + 6 = 10
Toss 4: 4 + 4 = 8
...
Toss 98: 5 + 2 = 7
Toss 99: 5 + 2 = 7
```

Now we want to count occurrences of each possible outcome. Let's use an array of integers. We will start with an array of zeros. And, for example, when we roll a 3, we'll add 1 to item 3 in the array. (We can never roll a zero or a one, so those two entries will always be zero.)

```
import random

roll_count = 100

# Make an array containing 13 zeros
counts = [0] * 13

for i in range(roll_count):
    a = random.randrange(6) + 1
    b = random.randrange(6) + 1
    roll = a + b
    print(f"Toss i: a + b = roll")

    # Increment the count for roll
    counts[roll] += 1

print(f"Counts: counts")
```

When you run this, at the end you will see a count for each possible outcome :

```
...
Toss 98: 3 + 2 = 5
Toss 99: 6 + 1 = 7
Counts: [0, 0, 2, 6, 16, 11, 13, 14, 11, 11, 6, 9, 1]
```

What was the count that we expected? For example, we expected to see a 2 about once every 36 rolls, right? It might be nice to compare our count to what we expected. Add a few more lines, and we are going to increase the number of rolls. You will probably want to delete the line that prints each roll separately:

```
import random

# Can't ever be 0 or 1
p = [0.0, 0.0, 1/36, 1/18, 1/12, 1/9, 5/36, 1/6, 5/36, 1/9, 1/12, 1/18, 1/36]
roll_count = 1000

# Make an array containing 13 zeros
counts = [0] * 13

for i in range(roll_count):
    a = random.randrange(6) + 1
    b = random.randrange(6) + 1
    roll = a + b

    # Increment the count for roll
    counts[roll] += 1

for i in range(2,13):
    print(f"i appeared counts[i] times, expected p[i] * roll_count:.1f")
```

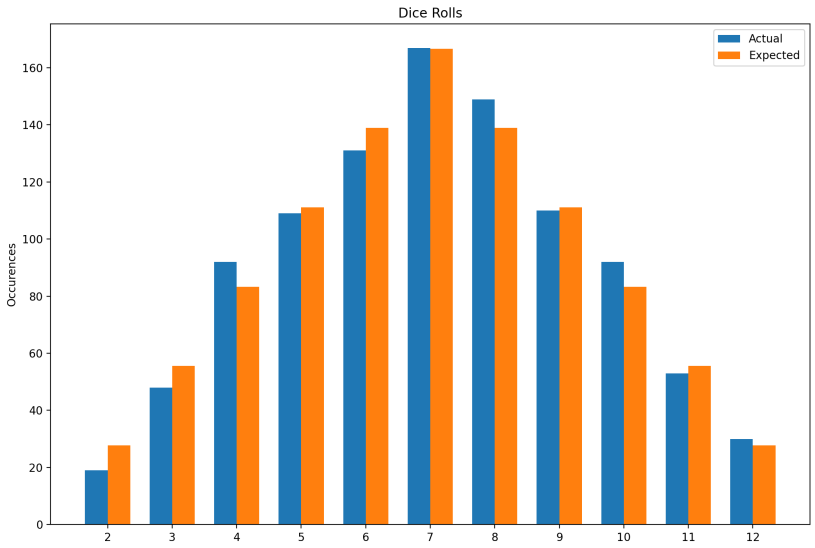
Now you should see something like:

```
2 appeared 39 times, expected 27.8
3 appeared 55 times, expected 55.6
4 appeared 84 times, expected 83.3
5 appeared 110 times, expected 111.1
6 appeared 160 times, expected 138.9
7 appeared 176 times, expected 166.7
8 appeared 124 times, expected 138.9
9 appeared 93 times, expected 111.1
10 appeared 87 times, expected 83.3
11 appeared 49 times, expected 55.6
12 appeared 23 times, expected 27.8
```

Whenever you are dealing with random numbers, the outcome will seldom be *exactly* what you expected. In this case, however, you should see that your predictions are pretty close.

1.4.1 Making a bar graph

A bar graph is a nice way to look at quantities like this. Let's make a bar graph that shows the actual count and the expected count:



We need to describe the set of rectangles, to do this we will loop through each possible roll (2 - 12) and put data in four lists for each:

```
import random
import matplotlib.pyplot as plt

# Can't ever be 0 or 1
p = [0.0, 0.0, 1/36, 1/18, 1/12, 1/9, 5/36, 1/6, 5/36, 1/9, 1/12, 1/18, 1/36]
roll_count = 1000

# Make an array containing 13 zeros
counts = [0] * 13

for i in range(roll_count):
    a = random.randrange(6) + 1
    b = random.randrange(6) + 1
    roll = a + b

    # Increment the count for roll
    counts[roll] += 1
```

```
# Gather data for bar chart
bar_width = 0.35
expected = []
actual_starts = []
expected_starts = []
labels = []
actual = []
for i in range(2,13):
    expected.append(p[i] * roll_count)
    actual.append(counts[i])
    actual_starts.append(i - bar_width/2)
    expected_starts.append(i + bar_width/2)
    labels.append(i)

fig, ax = plt.subplots()

# Create the bars
ax.bar(actual_starts, actual, bar_width, label='Actual')
ax.bar(expected_starts, expected, bar_width, label='Expected')
ax.set_xticks(labels)

# Provide labels
ax.set_ylabel('Occurences')
ax.set_title('Dice Rolls')
ax.legend()
plt.show()
```

Chapter 2

Beginning Combinatorics

Discrete probability problems often include some counting. For example, we figured out that there were 36 different ways the two dice, but all of them summed to some number 2 through 12. How many different ways could three 8-sided dice come up? We would need to count them, right? As the numbers get big we will need some tricks so we don't need to write them all down and count them one-by-one.

The branch of mathematics that focuses on tricks for counting is called *combinatorics*.

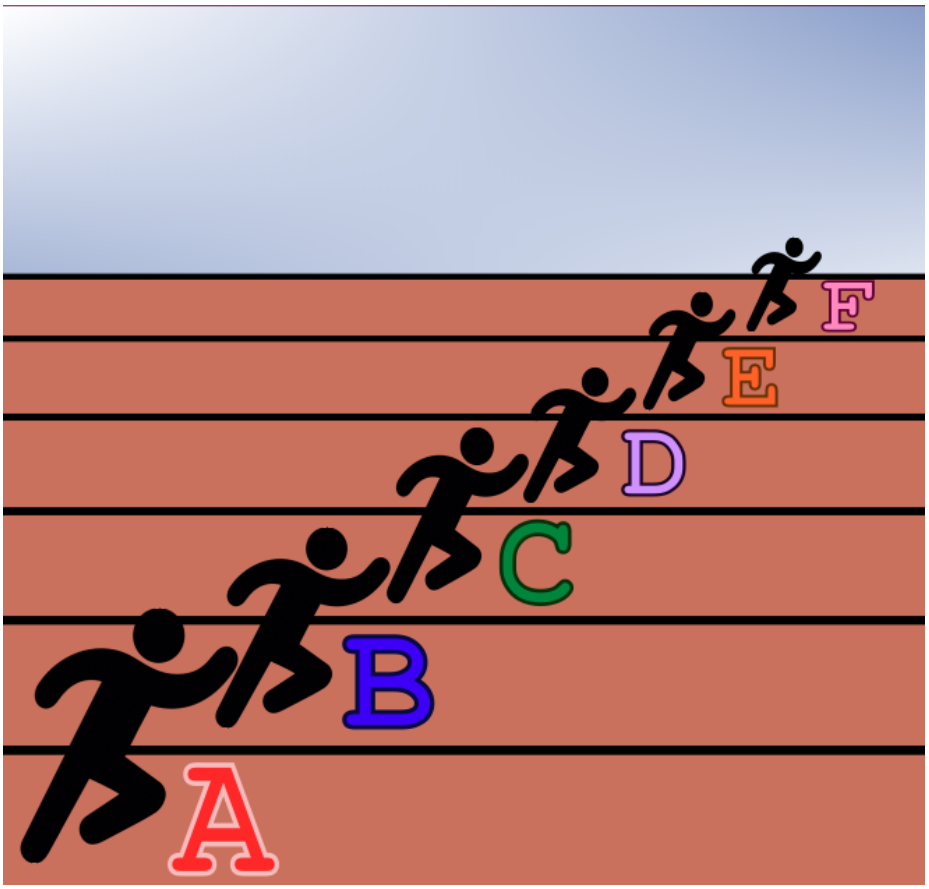
How can we be sure that there were 36 different configurations for the two 6-sided dice? The first die could have come up as any one of six numbers. For each of those, the second could have come up with any one of six numbers. Thus, the number of possibilities is $6 \times 6 = 36$.

How many different configurations for 3 8-sided dice? $8 \times 8 \times 8 = 8^3 = 512$.

What about seven dice, each with 20 sides? There would be $20^7 = 1,280,000,000$ configurations. See, aren't you glad we don't need to write them all down?

Now, let's say that six people (Anne, Brock, Carl, Dev, Edgar, and Fred) are going to run

a race. You have to make a plaque that says who won first place, who won second place, and who won third. If you want to get all the possible plaques created beforehand, and just pull the right one out as soon as the race ends, how many plaques would you need to get engraved?



In this case, once someone has been given first place, they can't win second or third place. Thus, any of the 6 people can come in first, but once you have engraved that person's name on the plaque, there are only 5 people whose names can appear in second place. Once you have engraved that name, there are only 4 people whose names can appear in third place. Thus, you would get $6 \times 5 \times 4 = 120$ plaques engraved.

What if the plaque includes all 6 places? Then you would need $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$ plaques engraved. We use this process often enough that we gave it a name. We say "I need 6 factorial plaques engraved." When we write a factorial, we use an exclamation point:

$$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$$

We use the word "permutation" to mean a particular ordering. This rule says n items can

be ordered in $n!$ ways. Thus mathematicians actually say "If you have a list of n items then we can generate $n!$ different permutations of those items".

In Python, there is a `factorial` function in the math library:

```
> python3
>>> import math
>>> math.factorial(6)
720
```

Handy, right? Now you don't need to write a loop to calculate factorials.

Remember when we only wanted the first three names on the plaque? We can do that problem using factorials:

$$6 \times 5 \times 4 = \frac{6 \times 5 \times 4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = \frac{6!}{3!}$$

This formulation makes it easy to figure out on any calculator with a "!" button.

The rule on this is to fill m positions from n items, it can be done this many ways:

$$\frac{n!}{(n-m)!}$$

2.0.1 Choose

Let's say that there are 12 kids in a classroom, and you need a team of 4 to wipe down the desks. How many different possible teams are there? You know that if you were giving out four different positions (Like the race gave out 1st, 2nd, and 3rd), the answer would be $12 \times 11 \times 10$ or $12!/(12-4)!$.

However, once we pick the 4 people, we don't care what order they are in, right? In this problem, the team "Anne, Brad, Carl, and Don" is the same as the team "Carl, Don, Brad, and Anne".

Thus, the quantity $12!/(12-4)!$ is many times too large because it counts each permutation separately. To get the right number, we just divide this by the number of possible permuations for a group of four people: $4!$

That gets us our answer: How many different teams of four can be chosen from 12 people?

$$\frac{12!}{(12-4)!4!} = 495$$

In combinatorics, we use this quantity a lot, so we have given it a name: *choose*

We have also given it a notation. "12 choose 4" is written like this:

$$\binom{12}{4}$$

Python has the `math.comb` function:

```
> python3
>>> import math
>>> comb(12, 4)
495
```

Chapter 3

Permutations and Sorting

In the previous chapter, we talked about permutations. If you have a list of three letters, like $[a, b, c, d]$, you can rearrange them in $4!$ ways:

a,b,c,d	a,b,d,c	a, d, b, c	a, d, c, b	a, c, b, d	a, c, d, b
b,a,c,d	b,a,d,c	b, d, a, c	b, d, c, a	b, c, a, d	b, c, d, a
c,b,a,d	c,b,d,a	c, d, b, a	c, d, a, b	c, a, b, d	c, a, d, b
d,b,c,a	d,b,a,c	d, a, b, c	d, a, c, b	d, c, b, a	d, c, a, b

You can make Python generate all the permutations for you:

```
from itertools import permutations
all_permutations = permutations(('a', 'b', 'c', 'd'))
for p in all_permutations:
    print(p)
```

3.1 Notation

How do we define or write down a single permutation? You could say something like "Swap the first and second items and swap the third and fourth items." However, that gets pretty difficult to read. So we usually write a permutation as two lines: the first line is before the permutation and the second line is after. Like this:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

And we can assign permutations to variables. For example, if we wanted the variable A to represent "swapping the first and second item", we would write this:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix}$$

And if we wanted B to represent "swapping the third and fourth item", we would write:

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 4 & 3 \end{pmatrix}$$

Now, we can *compose* permutations together. For example, we might say:

$$B \circ A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

That is if we have the list $[a, b, c, d]$ and we apply permutation A and then permutation B, we get $[b, a, d, c]$.

Important: Note that permutations are applied from right to left. $B \circ A$ means "Applying A and then B." Why does this matter? Permutations are not necessarily commutative. That is, if you have two permutations S and T, $S \circ T$ is not always the same as $T \circ S$.

Also, note that "don't change anything" is a permutation. We call it *the identity permutation*. If you have four items, the identity permutation would be written:

$$I = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

(We use a capital "I" for the identity.)

3.1.1 Challenge

Find an example of two permutations S and T such that $S \circ T$ does not equal $T \circ S$.

3.2 Sorting in Python

One of the common forms of permutation in software is sorting. Sorting is putting data in a particular order. For example, in Python, if you had a list of numbers, you can sort it in ascending order like this:

```
my_grades = [92, 87, 76, 99, 91, 93]
grades_worst_to_best = sorted(my_grades)
```

Do you want to sort backwards?

```
my_grades = [92, 87, 76, 99, 91, 93]
grades_best_to_worst = sorted(my_grades, reverse=True)
```

Note that `sorted` makes a new list with the correct order. If you want to sort the array in place, you can use the `sort` method:

```
my_grades = [92, 87, 76, 99, 91, 93]
my_grades.sort(reverse=True)
```

3.3 Inverses

Think for a second about this permutation:

$$S = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$$

You could say this permutation shuffles a list a bit. What is its inverse? That is, what is the permutation that unshuffles the items back to where they were originally?

$$S^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$$

That is, the original moved an item in the first spot to the third spot. The inverse must move whatever was in the third spot back to the first spot.

(Notation note: Because in multiplication, $b \times b^{-1} = 1$, we use "to the negative one" to indicate inverses in lots of places.)

Mechanically, how do you find the inverse? Flip the rows, and then sort the columns using the top number:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix} \text{ flip } \rightarrow \begin{pmatrix} 3 & 4 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \text{ sort } \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$$

Let's say you have two permutations A and B. Permuting by B and then A would look like this:

$$C = A \circ B$$

If you know A^{-1} and B^{-1}, what is C^{-1}? You would undo-A and then undo-B, so

$$C^{-1} = B^{-1} \circ A^{-1}$$

3.4 Cycles

Here is a permutation:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix}$$

When this is applied, whatever is at 1 gets moved to 2, 2 gets moved to 4, and 4 gets moved to 1. That is a *cycle*: $1 \rightarrow 2 \rightarrow 4$ and then it goes back to 1. It involves three locations, so we say it is a *3-cycle*.

There is another cycle in this permutation: $3 \rightarrow 5$ and then it goes back to 3.

Because these cycles share no members, we say the cycles are *disjoint*.

Every permutation can be broken down into a collection of disjoint cycles.

$$T = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix} = (1 \rightarrow 2 \rightarrow 4)(3 \rightarrow 5)$$

The first handy thing about this notation is that it makes it easy for us to describe the inverse: we just run the cycles backward:

$$T^{-1} = (4 \to 2 \to 1)(5 \to 3)$$

Starting with the list $[a, b, c, d, e]$, lets repeatedly apply the permutation T

Initial	a, b, c, d, e
T applied	d, a, e, b, c
$T \circ T$ applied	b, d, c, a, e
$T \circ T \circ T$ applied	a, b, e, d, c
$T \circ T \circ T \circ T$ applied	d, a, c, b, e
$T \circ T \circ T \circ T \circ T$ applied	b, d, e, a, c
$T \circ T \circ T \circ T \circ T \circ T$ applied	a, b, c, d, e

This permutation, results in six combinations, and then it loops back on itself. The number of combinations is the least common multiple of all the cycles. In this case, there is a 3-cycle and a 2-cycle. The least common multiple of 2 and 3 is 6.

Appendix A

Answers to Exercises

Answer to Exercise 1 (on page 5)

probability of all 5's $= \frac{1}{6} \times \frac{1}{6} \times \frac{1}{6} = \left(\frac{1}{6}\right)^3 = \frac{1}{216} \approx 0.0046$

Answer to Exercise 1 (on page 5)

probability of at least one heads = 1.0 - probability of all tails $= 1.0 - \left(\frac{1}{2}\right)^5 = 1.0 - \frac{1}{32} = \frac{31}{32} = \approx 0.97$

Index