Counting Worlds:
From Usernames to Universes

Jeremy Evert

November 11, 2025

Contents

Chapter 1

Welcome to *Counting Worlds*

1.1 A Story About Too Many Possibilities

Imagine this:

- You open a new game.
- You have to choose a character: race, class, hairstyle, outfit, special skill.
- Five minutes later, you are still on the character creation screen.

It feels like there are *infinitely many* options. In reality, there is a large but finite *universe of possibilities*. This chapter—and this whole mini-unit—is about learning how to *count* those universes.

We are going to ask questions like:

- How many ways can we seat six friends in a row of chairs?
- How many usernames can a website support under a given naming rule?
- How many sundaes can we build from a limited supply of scoops and flavors?
- How often should a particular dice pattern show up if our math is right?
- How big is the universe of characters, decks, or skill builds in a game?

Instead of guessing, we will use discrete mathematics and a bit of Python to get real answers. Along the way, we will see that:

> *Counting is how we tame spaces that feel infinite.*

1.2 What You Will Be Able to Do

By the end of this counting unit, you should be able to:

Mathematical skills

- Use the product principle ("AND" means multiply) and the sum principle ("either/or" means add, when choices are disjoint) in real problems.
- Work with *permutations* (where order matters) using factorials.
- Work with *combinations* (where order does not matter) using binomial coefficients $\binom{n}{k}$.
- Use the *stars and bars* technique to count ways of distributing identical items (like scoops, points, or coins) into bins.
- Connect counting to *probability* in simple finite situations.

Computational skills

- Use Python as an "experimental math lab":
  - generate all outcomes for small cases;
  - check whether a formula seems to be right;
  - estimate probabilities via Monte Carlo simulation.
- Translate informal stories (about games, food, or chaos) into:
  - precise combinatorial models, and
  - short, readable Python scripts.

Design and creativity

- Design your own small "universe" (a game, system, or scenario).
- Identify where permutations, combinations, and stars-and-bars appear inside that universe.
- Justify your counting formulas in words, not just symbols.

1.3 How This Mini-Book Works

Each chapter follows the same basic rhythm:

1. **Start with a story.** We begin with something that feels familiar: seating friends, building sundaes, creating usernames, rolling dice, or designing a game world.
2. **Extract the structure.** We strip away the flavor and look at the underlying combinatorial skeleton: choices, constraints, order vs. no order, repetition vs. no repetition.
3. **Do the math.** We use the tools of discrete mathematics to turn the story into symbols and formulas and compute exact counts.
4. **Check or explore with Python.** For small versions of the problem, we let a Python script:
   - generate the whole universe explicitly,
   - count how many outcomes have a certain property,

- approximate probabilities with random trials.

5. **Reflect and remix.** At the end of the chapter, we check understanding with short exercises and a brief "podcast" conversation that ties the ideas back to the story.

You do *not* need to be a Python expert. The code will be short and focused, and you can treat it as executable pseudocode if you are still learning the language.

1.4 Python as Our Counting Laboratory

Here is roughly how Python shows up throughout this unit:

- We use `itertools` to generate permutations, combinations, and simple products (like all possible usernames of a given form).
- We use the `math` module for things like `math.factorial` and `math.comb`.
- We use the `random` module to simulate coin flips, dice rolls, and other random experiments.

Most scripts will follow this template:

1. Set up a small, concrete version of the problem.
2. Compute a theoretical count or probability using a formula.
3. Use Python to:
   - either enumerate all possibilities, or
   - run a large number of random trials.
4. Compare the result from Python with the formula.

You will be encouraged to modify the scripts: change parameters, add constraints, or invent your own stories that use the same combinatorial ideas.

1.5 Roadmap for the Five Chapters

Here is the plan for the rest of this counting unit:

Chapter 1: Seating the Party at the Round Table

We start with *factorials* and *permutations*. You will:

- Count the number of ways to seat people in a row.
- Handle restrictions (must sit together, must not sit together, fixed seats).
- Use Python to brute-force small seating problems and check your formulas.

Chapter 2: Usernames, License Plates, and Other Noisy Strings

We move to *strings with repetition*. You will:

- Model usernames, license plates, and PINs as strings from an alphabet.
- Use the product principle to count how many such strings exist.
- Compare different username rules and discuss which ones create larger (or smaller) universes of names.

Chapter 3: How Many Sundaes Can We Build?

We introduce *stars and bars*. You will:

- Count ways to distribute scoops among flavors or points among skills.
- Visualize stars-and-bars as a picture and connect it to a formula.
- Use Python to generate all small distributions and verify your counts.

Chapter 4: Roll the Dice, Check the Math

We tie counting to *probability* and *simulation*. You will:

- Compute exact probabilities using combinatorial reasoning.
- Run Monte Carlo simulations in Python to estimate those probabilities.
- Watch simulated frequencies drift toward theoretical values as the number of trials increases.

Chapter 5: Design Your Own Universe

Finally, you become the world-builder. You will:

- Design a small universe of your choice (game, system, or scenario).
- Identify at least one permutation, one combination, and one stars-and-bars situation inside it.
- Analyze your universe with formulas *and* Python, then present your findings.

Podcast: Episode 0 — Trailer for the Counting Worlds

At the end of this intro, there is a short podcast episode that you can listen to while walking, commuting, or setting up your Python environment. The episode can follow this rough script:

- Introduce the "cast" of the unit (for example, recurring student characters or narrators).
- Share one real-life story where counting matters (picking teams, building a schedule, designing a game).
- Tease each chapter in one sentence:

  - "First, we figure out how many ways to seat a chaotic friend group."
  - "Then, we see how many usernames and license plates fit in the system."
  - "Next, we over-engineer sundae bars and skill trees."
  - "After that, we roll dice to see if our math holds up."
  - "Finally, *you* design your own universe and count it."

- Close with a simple challenge:

  Before Chapter 1, try to guess: *How many different outfits can you build from your own closet?*

This podcast is not graded; it is here to set the tone, tell stories, and give you a human voice walking you into the math.

Chapter 2

Seating the Party at the Round Table

2.1 Story Hook: Dinner Disaster

You are in charge of a big group dinner.

There are six friends and six chairs in a row. Everyone has Opinions:

- Amina *must* sit on an end.
- Jake refuses to sit next to Zahra.
- Lin and Zahra are best friends and want to sit together.

The host asks you a simple question:

"How many different seating charts are possible?"

At first, this seems like a small thing. But as the number of people grows, the number of possible seatings explodes. By the end of this chapter you will know:

- how to count all possible seatings when everyone is distinct;
- how to handle simple constraints (must sit together / apart / on an end);
- how to use Python to *check* your counting on small examples.

2.2 Warm-Up: Listing All Seatings for Three Friends

Let us start very small. Suppose we only have three people: Amina (A), Lin (L), and Jake (J), and three chairs in a row.

Try it yourself

1. Draw three boxes to represent the chairs.
2. List every possible way to place A, L, and J into those chairs.

If you do this carefully, you should find these six arrangements:

$$\text{ALJ},\ \text{AJL},\ \text{LAJ},\ \text{LJA},\ \text{JAL},\ \text{JLA}.$$

There are 6 different seatings. This is already a lot for just three people! Soon we will see how this connects to the factorial function and why three people give six seatings, four people give twenty-four seatings, and so on.

2.3 Core Ideas: Factorials and Linear Permutations

2.3.1 The factorial function

The *factorial* of a positive integer n is written $n!$ and defined as

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1,$$

with the special convention that $0! = 1$.

Some small values:

$$1! = 1, \quad 2! = 2, \quad 3! = 6, \quad 4! = 24, \quad 5! = 120.$$

2.3.2 Factorials as seatings

We can interpret $n!$ as the number of ways to seat n distinct people in n distinct chairs in a row.

Why?

- For the first chair, we can choose any of the n people.
- For the second chair, we have $n-1$ people left.
- For the third chair, we have $n-2$ people left.
- ...
- For the last chair, we have 1 person left.

By the *product principle*, the total number of seatings is

$$n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1 = n!.$$

This explains why:

- $3! = 6$ seatings for A, L, J (which we listed),
- $4! = 24$ seatings for four distinct people,
- $6! = 720$ seatings for six distinct people.

Already at six people, 720 is a lot of possible seating charts. No human host is going to check all of those by hand.

2.3.3 Permutations of a set

If we have a set S of n distinct elements, any ordering of the elements of S in a row is called a *permutation* of S. The discussion above says:

> The number of permutations of a set of size n is $n!$.

This is one of the most basic counting facts in discrete mathematics: whenever you are arranging n distinct things in a line and every order is allowed, the answer is $n!$.

2.4 Restricted Seatings: Adding Drama

Real dinners have rules. Let us see how a few common restrictions change the counting. Throughout this section, think of six seats in a row and six friends: Amina (A), Lin (L), Zahra (Z), Jake (J), plus two more friends, M and N.

2.4.1 Example 1: One person in a fixed seat

Suppose Amina *must* sit in the first chair.

- We no longer have a choice for the first chair: it is always A.
- For the remaining five chairs, we can seat the other five friends in any order.

So the total number of seatings is simply

$$5!$$

because we are only permuting the remaining five people.

In general, if one person has a fixed location and the others are free, the number of seatings is $(n-1)!$.

2.4.2 Example 2: Two people sit together as a block

Now suppose Lin and Zahra insist on sitting next to each other.

A standard trick is to treat the pair (LZ) as a single "block."

- First, create a new list of "objects": the block (LZ) plus the other four people A, J, M, N. That gives 5 objects to arrange.
- The number of ways to arrange these 5 objects is 5!.
- Inside the block, Lin and Zahra can sit in two orders: LZ or ZL.

By the product principle:

$$\text{number of seatings with L,Z together} = 5! \cdot 2.$$

This idea appears over and over: when some group must stay together, we often treat that group as a single unit, count arrangements of the units, then multiply by the internal arrangements of each unit.

2.4.3 Example 3: Two people must not sit together

Now suppose Jake and Zahra *refuse* to sit next to each other. How many seatings avoid J and Z being adjacent?

A common strategy is:

1. Count all possible seatings with no restriction: 6!.
2. Count how many seatings have J and Z together (as a block).
3. Subtract: (all seatings) − (bad seatings).

We already know from the previous example that the number of seatings with J and Z together is $5! \cdot 2$ (treat JZ as a block, and swap inside the block). Therefore:

$$\text{seatings with J and Z not adjacent} = 6! - 5! \cdot 2.$$

This pattern—*count everything, then subtract the bad cases*—is a very powerful idea in counting. We will see it again later in the unit with more complicated sets of "bad" outcomes.

2.5 Python Lab: Brute-Forcing the Seating

Mathematics gives us formulas, but Python lets us *test* those formulas on small examples and explore different constraints without doing all the bookkeeping by hand.

In this lab you will write or modify a script (for example, `round_table.py`) with the following goals.

Step 1: Generate all seatings

1. Create a list of names, such as `["Amina", "Lin", "Zahra", "Jake"]`.
2. Use `itertools.permutations` to generate all possible orderings of the list.
3. Confirm that the number of permutations matches $n!$ for your n.

A short code sketch might look like this:

```
import itertools
import math

friends = ["A", "L", "Z", "J"]
perms = list(itertools.permutations(friends))

print("Number of permutations:", len(perms))
print("Should be:", math.factorial(len(friends)))
```

Step 2: Count seatings with a fixed person at one end

Modify your code so that it counts only those permutations where Amina sits in the first chair.

- Compare the Python count to the formula $(n-1)!$.
- Try both ends (first or last) and see what happens.

Step 3: Count seatings where two friends sit together

Choose two friends (for example, Lin and Zahra) and:

- count how many permutations have them sitting in adjacent seats;
- compare to the "block" formula $5! \cdot 2$ (for $n = 6$);
- experiment with different values of n and positions.

Step 4: Count seatings where two friends do *not* sit together

Finally, let Python estimate how many permutations avoid J and Z being neighbors.

- Compute the number directly by checking every permutation.
- Compare with the subtraction formula $6! - 5! \cdot 2$.
- Try other pairs of friends and see if the pattern holds.

The goal is not to write huge programs, but to use short scripts as mirrors that reflect the combinatorial structure we discovered on paper.

2.6 Practice and Extensions

Here are some practice problems you might see after reading this chapter.

Basic practice

1. Explain in words what 5! means in the context of seating five distinct friends in five chairs.
2. Compute 6! and describe a real-life scenario where 6! is the correct answer.

Medium spice

1. Six friends are to be seated in a row. Two of them insist on sitting together. How many seatings are possible?
2. Six friends are to be seated in a row. Two of them refuse to sit next to each other. How many seatings are possible?

Extra spicy (optional)

1. Circular table variation: six friends sit around a round table. Two seatings that can be rotated into each other are considered the same. How many distinct seatings are possible?
2. Modify your Python script to treat rotations as identical and test your answer on small values of n.

Podcast: Episode 1 — The Seating Chart

At the end of this chapter, there is a short podcast episode. A possible outline for the episode:

- The characters are trying to organize a group dinner.
- They first try to list out seatings for a small group and quickly get overwhelmed as the group grows.
- Someone introduces the idea of $n!$ and the "product of choices" story: first seat, second seat, and so on.

- They play with the idea of treating two friends as a "block" and of subtracting the bad arrangements.
- They mention that Python helped them check their answers by generating all seatings for small examples.
- They end with a teaser:

> "If we can count seatings, can we also count usernames? Next time: gamer tags, license plates, and the size of the internet."

Chapter 3

Usernames, License Plates, and Other Noisy Strings

Chapter 4

How Many Sundaes Can We Build?

4.1 Story Hook: Infinite Ice Cream Bar

4.2 Stars and Bars: Distributing Identical Objects

4.3 Variants: Minimums and Caps

4.4 Python Lab: Enumerating Sundaes

4.5 Practice: Scoops, Spells, and Skill Trees

Podcast: Episode 3 – Sundae Architect

Chapter 5

Roll the Dice, Check the Math

Chapter 6

Design Your Own Universe

6.1 Project Brief

6.2 Planning the Universe

6.3 Mathematical Analysis

6.4 Python Component

6.5 Presentations and Reflection

Podcast: Episode 5 – Universe Builders