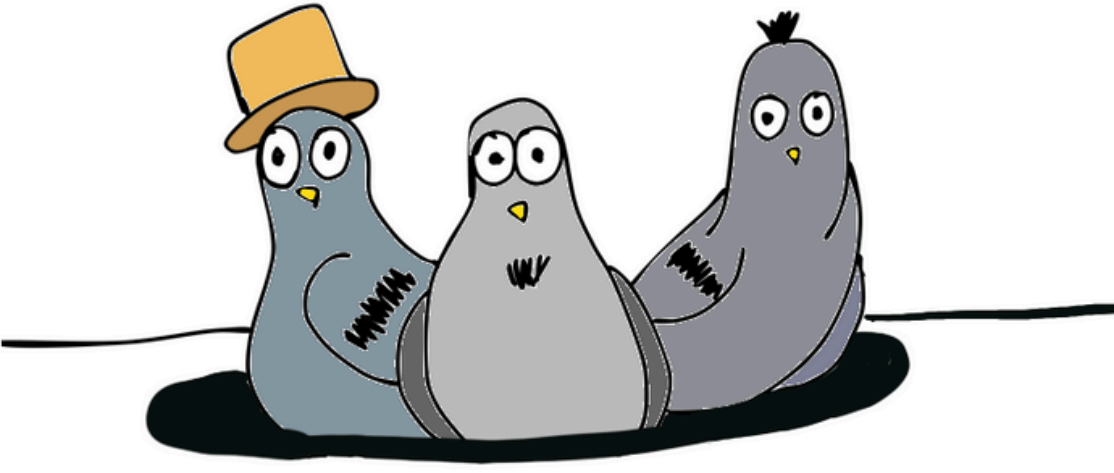
# Pigeons in Holes

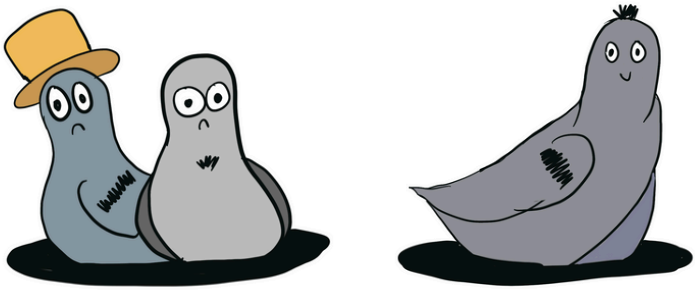
By Alisha Manocha



# What is the Pigeonhole Principle?

The Pigeonhole Principle is a combinatorial principle that is quite intuitive but can lead to some interesting, surprising results!

Say we have 3 pigeons who live in 2 holes.
Obviously, two of our pigeons have to sacrifice their privacy.

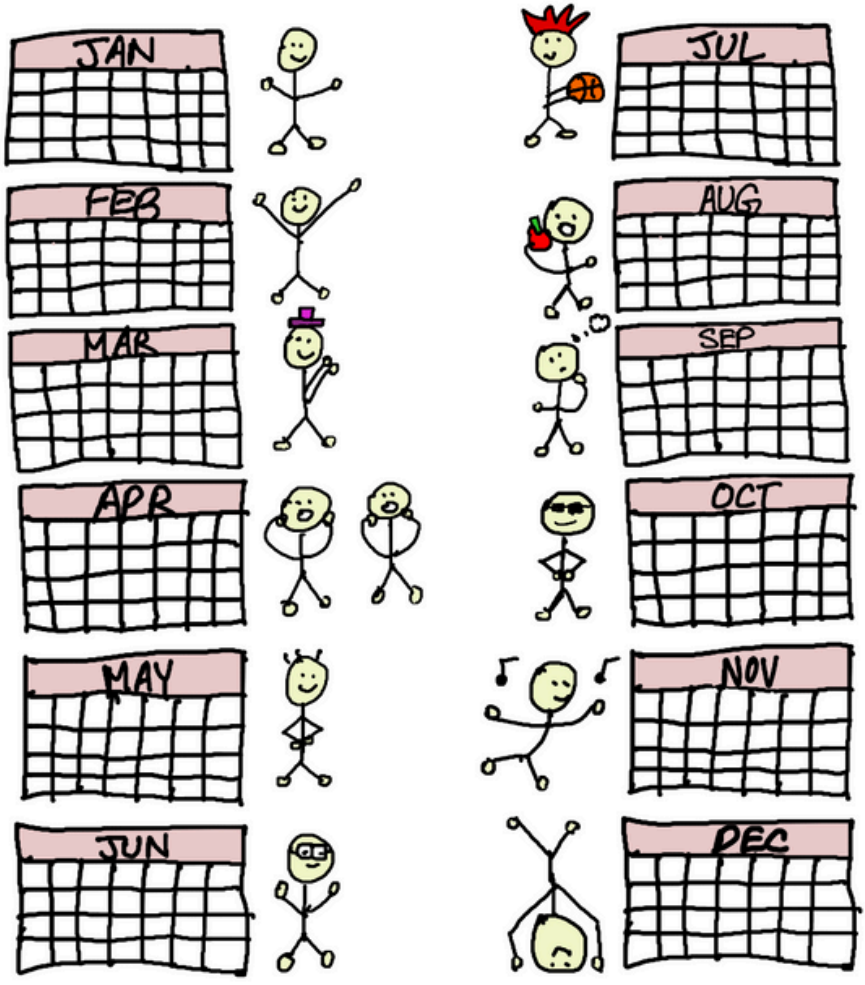


Similarly, if we had 2 pigeons and 3 holes, we could guarantee that at least one of the holes was empty.

This can be generalized to n items that we place in k boxes:
If $n > k$, at least one box contains more than one item.
If $n < k$, at least one box contains no items.

# Some Examples

In a room of 13 people, at least two must share the same birthday month (since 13 > 12).



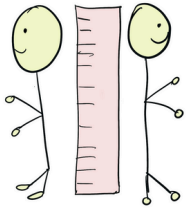

Our pigeons here are the people's birthdays, and the holes are the months.

Let's look at some more interesting ones, though.

In a town of 3000 people, at least two people are the same height in millimeters.

The tallest person ever recorded stood at 8 ft 11.1 in (272 cm), or 2720 mm. Thus, assuming that no person in this town beats that record, there are only 2720 possible heights in millimeters.

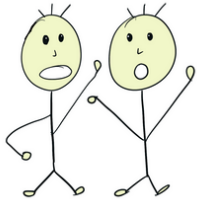
Then, since 3000 > 2720, at least two of the 3000 residents share a height.



There are at least two people living in Los Angeles with the same number of hairs on their heads.

The average person has between 90,000 and 150,000 hairs on their head, but just to be safe, let's say the maximum number of hairs on the human head is one million. As of 2023, the population of LA was 3.821 million.

So, the claim follows from the fact that this population size exceeds our maximum number of hairs that a human can have.

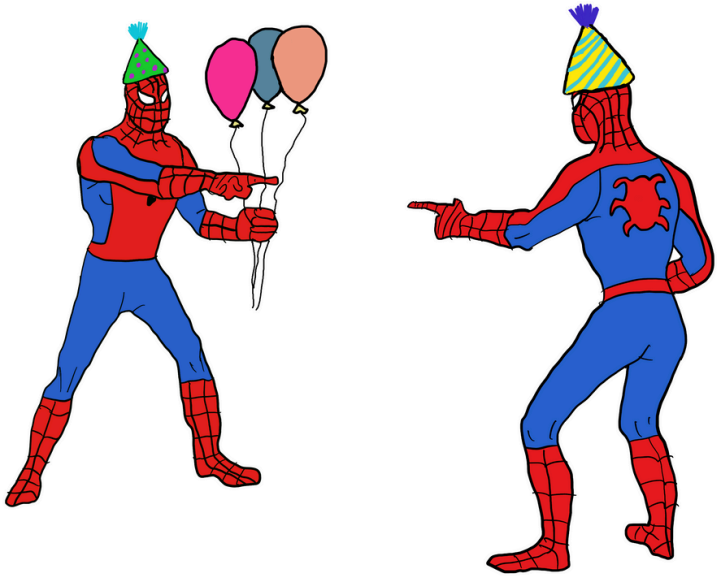


# Ok, this is cool and all, but why does it matter?

Well, the pigeonhole principle brings up some interesting consequences on physical limitations and uniqueness.

One relevant example is the birthday paradox and its applications to cryptography.

It answers the question: How many people do you need to put in a room to have a 50% chance of at least two of them sharing a birthday?



It seems counter-intuitive, but the answer is just 23.

In fact, only 75 people are needed to raise this probability to 99.97%!

Finding the probability of a birthday pair in n people looks something like this:

$$
\begin{aligned}
P(\text{pair in } n \text{ people}) &= 1 - P(\text{all } n \text{ people have different birthdays}) \\
&= 1 - \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\left(\frac{363}{365}\right)\cdots\left(\frac{365-n+1}{365}\right) \\
&= 1 - \left(\frac{1}{365}\right)^n \left(\frac{365!}{(365-n)!}\right) \\
&\approx 1 - e^{-\frac{n^2}{730}} \quad (\text{using a Taylor approximation})
\end{aligned}
$$

So, a 50% probability of this occurring is achieved if

$$
\begin{aligned}
0.5 &\leq 1 - e^{\frac{-n^2}{730}} \\
\Rightarrow n &\geq \sqrt{-730 \ln 0.5} \\
&\geq 22.494
\end{aligned}
$$

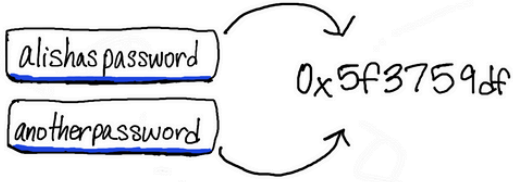
Thus, only 23 people are needed to have the odds in favor of finding two birthday buddies!

While this example involves probability, we see the same underlying reasoning as that of the pigeonhole principle — when you distribute people to a limited number of birthday slots, shared birthdays become inevitable as the number of people grows.

This tells us that the same thing happens for other kinds of mappings, and in cases where we would desire uniqueness, this can cause some problems! Let's briefly examine one example.

A cryptographic hash function maps inputs of all sizes to fixed-size outputs (hash values). Let's consider a hash function that maps input passwords to 32-bit hash values. We then have $2^{32}$, or about 4.29 billion hashes.

A hashing "collision" occurs when two passwords map to the same hash.




This would be bad since someone could seemingly access my resources with the wrong password.

A good hashing algorithm will do its best to avoid collisions, but as we have seen, past a certain number of passwords existing in the system, it becomes more and more likely that collisions will occur.

If someone were to try to brute-force finding my password or even a password that collides with mine, it would simply be computationally infeasible. However, if they were just interested in finding a collision between any two passwords, the birthday paradox would work in their favor.

This is known as the "birthday attack," and in our 32-bit hash example, on average, only $n \geq \sqrt{-2^{33} \ln 0.5} \approx 77163$ guesses are needed to find a collision!

A common example is if a malicious actor can find two contract documents — one good and one bad — with the same hash, they could get your signature on the good one and swap it out for the bad one.

# References


Bellore, Arthur. "Birthday Attacks, Collisions, and Password Strength." Auth0, 23 Mar. 2021, auth0.com/blog/birthday-attacks-collisions-and-password-strength/.

Hammack, Richard. Book of Proof. 3.4 ed., 2018.

"Understanding the Birthday Paradox." BetterExplained, betterexplained.com/articles/understanding-the-birthday-paradox/.


**An interesting read if you would like to learn more about Birthday Attacks!**


"Unpacking the Birthday Attack in Hashing." Blue Goat Cyber, bluegoatcyber.com/blog/unpacking-the-birthday-attack-in-hashing/.

Ali, Ali, and Lindsey Kuper. Communicating Chorrectly with a Choreography, decomposition.al/zines/communicating-chorrectly-zine.pdf.


**A cool zine that inspired this one!**