PROFESSOR: The topic of counting, or combinatorics, is an important one in a number of different disciplines, and notably in computer science.

So the origins of the combinatorics and counting are a little bit disreputable. They come, historically, out of people studying gambling and trying to calculate odds and the fraction of times that various events occur to know what kind of bets to make on them.

So a typical kind of question would be-- if you know how poker works, there are various classifications of five-card hands in poker. And you might ask, what fraction of all possible five-card poker hands translate into being a pair of jacks? And basically, this fraction of total poker hands which fit into the category "a pair of jacks" is the probability of a pair of jacks. So counting in gambling is one fundamental place where it really comes up. And historically, that's where a lot of combinatorics begins.

Related to that is counting in games. When you're trying to write, for example, a computer program to play chess or checkers or so on, one of the aspects of it is getting a sense of how much searching you have to do in order to look ahead to find good moves. And you wind up counting, from a given chess position, how many possible further positions can you get to a given number of moves.

A puzzle kind of question, in solving the Rubik's Cube toy, is how many different positions are there, and how many different positions can you get to from a given position?

Concretely, in computer science, it comes up in algorithms. It's often the case that an essential question is, how many operations does it take to do a manipulation on a data structure and to update it from one to another? For example, how many comparisons does it take to sort n numerical items? And typically, the count is n log n, proved as a number of operations that are both achievable and a lower bound.

One that we've seen when we looked at fast exponentiation is a question like, if you're trying to compute the nth power of a number d, how many multiplications does it take? It's roughly log n by using the iterated squaring method. And we want to be able to count that number of multiplications that a particular program uses to compute d to the n in the smallest number of multiplications you can get away with.

And a place where, again, counting and combinatorics becomes critical is for security and the issue of cryptography. If you're going to have security from passwords, there needs to be too large a space of passwords for an adversary to search through exhaustively and check them likewise. If you're doing encryption with some kind of secret key that enables you to read messages, you want to be sure that the space of possible keys is also way too large to search exhaustively to see what keys work.

So let's talk briefly now about the very basic counting methods, and two rules for counting things-- the most rudimentary of them, but in fact we'll get some mileage out of them.

So the first rule is called the Sum Rule, and it's completely straightforward and obvious, which is that if I have two sets, A and B, that do not overlap, then the number of elements in A union B is simply the number of elements in A plus the number of elements in B. And there's no issue proving that-- it's self-evident-- but let's do an example.

Suppose a class has 43 women and 54 men. How many people are in it? 43 plus 54 equals 97. This is implicitly assuming that there's no one whose sex is ambiguous and that there's no third sex, so that men and women are disjoint. The total number of students is the sum of the number of men and women.

Another one is there are 26 lowercase Roman letters and 26 uppercase Roman letters and 10 digits. And so there are 26 plus 26 plus 10 equals 62 characters in that repertoire of symbols.

The second rule is called the Product Rule, and just about as obvious. Suppose I have four boys and three girls. How many boy-girl couples can I assemble out of four boys and three girls? And the answer is, there are four ways to choose a boy, and for each of them, there are three ways to choose a girl. So there's 4 times 3, or 12, possible boy-girl girl couples in this setting.

More generally, if I have a set A of size m and a set B of size n, then A cross B-- remember, that's the set of ordered pairs where the first element is from A and the second element is from B-- the size of A cross B is-- the vertical bars, remember, mean size-- is equal to m times n.

So let's just do an example that illustrates it. Suppose that A is the set of four elements-- little a, b, c, and d-- and B is the set of three numbers, 1, 2, and 3. Then I can list A cross B in a nice orderly way, as a 4-by-3 matrix. But this is really meant to be just a list of elements, but

I'm organizing this way so the pattern is more apparent.

And for each element little a, I can pair it with each of the three elements in B. And for the second element, little b, in A, I can pair it with this three digits [? in A. ?] And c I can pair with three, and d I can pair with three. And that's where the 4 times 3 comes from, and more generally, the m times n comes from.

A useful immediate application of this is, how many binary strings are there? How many strings of zeros and ones are there of length 4? Well, the length for binary strings, it can be explained as well as the product of B times B times B times B. We're not writing parentheses here. It's B times B-- cross B.

So I'm thinking of a quadruple like this as being a pair whose first element is a triple. And a triple is a pair whose first element is a pair. And given that it doesn't really matter how you break it up, we just typically write it as B cross B cross B, and even abbreviate that as B to the fourth, where b is 0, 1.

And the Product Rule says that the size of this is the size of B times the size of B times the size of B times the size of B, or 2 to the fourth. So in general, if I look at strings of length n, whose elements are from an alphabet of size m, the total number of such strings is m to the n.