Fixed-length encodings work well when all the possible choices have the same information content, i.e., all the choices have an equal probability of occurring.

If those choices don't have the same information content, we can do better.

To see how, consider the expected length of an encoding, computed by considering each x_i to be encoded, and weighting the length of its encoding by p_i, the probability of its occurrence.

By "doing better" we mean that we can find encodings that have a shorter expected length than a fixed-length encoding.

Ideally we'd like the expected length of the encoding for the x_i to match the entropy H(X), which is the expected information content.

We know that if x_i has a higher probability (i.e., a larger p_i), that is has a smaller information content, so we'd like to use shorter encodings.

If x_i has a lower probability, then we'd use a longer encoding.

So we'll be constructing encodings where the x_i may have different length codes - we'll call these variable-length encodings.

Here's an example we've seen before.

There are four possible choices to encode (A, B, C, and D), each with the specified probability.

The table shows a suggested encoding where we've followed the advice from the previous slide: high-probability choices that convey little information (e.g., B) are given shorter encodings, while low-probability choices that convey more information (e.g., C or D) are given longer encodings.

Let's diagram this encoding as a binary tree.

Since the symbols all appear as the leaves of the tree, we can see that the encoding is unambiguous.

Let's try decoding the following encoded data.

We'll use the tree as follows: start at the root of the tree and use bits from the encoded data to traverse the tree as directed, stopping when we reach a leaf.

Starting at the root, the first encoded bit is 0, which takes us down the left branch to the leaf B. So B is the first symbol of the decoded data.

Starting at the root again, 1 takes us down the right branch, 0 the left branch from there, and 0 the left branch below that, arriving at the leaf C, the second symbol of the decoded data.

Continuing on: 11 gives us A, 0 decodes as B, 11 gives us A again, and, finally, 101 gives us D. The entire decoded message is "BCABAD".

The expected length of this encoding is easy to compute: the length of A's encoding (2 bits) times its probability, plus the length of B's encoding (1 bit) times 1/2, plus the contributions for C and D, each 3 times 1/12.

This adds up to 1 and 2/3 bits.

How did we do?

If we had used a fixed-length encoding for our four possible symbols, we'd have needed 2 bits each, so we'd need 2000 bits to encode 1000 symbols.

Using our variable-length encoding, the expected length for 1000 symbols would be 1667.

The lower bound on the number of bits needed to encode 1000 symbols is 1000 times the entropy H(X), which is 1626 bits, so the variable-length code got us closer to our goal, but not quite all the way there.

Could another variable-length encoding have done better?

In general, it would be nice to have a systematic way to generate the best-possible variable-length code, and that's the subject of the next video.