If the symbols we are trying to encode occur with equal probability (or if we have no a priori reason to believe otherwise), then we'll use a fixed-length encoding, where all leaves in the encoding's binary tree are the same distance from the root.

Fixed-length encodings have the advantage of supporting random access, where we can figure out the Nth symbol of the message by simply skipping over the required number of bits.

For example, in a message encoded using the fixed-length code shown here, if we wanted to determine the third symbol in the encoded message, we would skip the 4 bits used to encode the first two symbols and start decoding with the 5th bit of message.

Mr. Blue is telling us about the entropy for random variables that have N equally-probable outcomes.

In this case, each element of the sum in the entropy formula is simply (1/N)*log2(N), and, since there are N elements in the sequence, the resulting entropy is just log2(N).

Let's look at some simple examples.

In binary-coded decimal, each digit of a decimal number is encoded separately.

Since there are 10 different decimal digits, we'll need to use a 4-bit code to represent the 10 possible choices.

The associated entropy is log2(10), which is 3.322 bits.

We can see that our chosen encoding is inefficient in the sense that we'd use more than the minimum number of bits necessary to encode, say, a number with 1000 decimal digits: our encoding would use 4000 bits, although the entropy suggests we *might* be able to find a shorter encoding, say, 3400 bits, for messages of length 1000.

Another common encoding is ASCII, the code used to represent English text in computing and communication.

ASCII has 94 printing characters, so the associated entropy is log2(94) or 6.555 bits, so we would use 7 bits in our fixed-length encoding for each character.

One of the most important encodings is the one we use to represent numbers.

Let's start by thinking about a representation for unsigned integers, numbers starting at 0 and counting up from there.

Drawing on our experience with representing decimal numbers, i.e., representing numbers in "base 10" using the 10 decimal digits, our binary representation of numbers will use a "base 2" representation using the two binary digits.

The formula for converting an N-bit binary representation of a numeric value into the corresponding integer is shown below – just multiply each binary digit by its corresponding weight in the base-2 representation.

For example, here's a 12-bit binary number, with the weight of each binary digit shown above.

We can compute its value as 0*2^11 plus 1*2^10 plus 1*2^9, and so on.

Keeping only the non-zero terms and expanding the powers-of-two gives us the sum 1024 + 512 + 256 + 128 + 64 + 16 which, expressed in base-10, sums to the number 2000.

With this N-bit representation, the smallest number that can be represented is 0 (when all the binary digits are 0) and the largest number is 2^N – 1 (when all the binary digits are 1).

Many digital systems are designed to support operations on binary-encoded numbers of some fixed size, e.g., choosing a 32-bit or a 64-bit representation, which means that they would need multiple operations when dealing with numbers too large to be represented as a single 32-bit or 64-bit binary string.

Long strings of binary digits are tedious and error-prone to transcribe, so let's find a more convenient notation, ideally one where it will be easy to recover the original bit string without too many calculations.

A good choice is to use a representation based on a radix that's some higher power of 2, so each digit in our representation corresponds to some short contiguous string of binary bits.

A popular choice these days is a radix-16 representation, called hexadecimal or "hex" for short, where each group of 4 binary digits is represented using a single hex digit.

Since there are 16 possible combinations of 4 binary bits, we'll need 16 hexadecimal "digits": we'll borrow the ten digits "0" through "9" from the decimal representation, and then simply use the first six letters of the alphabet, "A" through "F", for the remaining digits.

The translation between 4-bit binary and hexadecimal is shown in the table to the left below.

To convert a binary number to "hex", group the binary digits into sets of 4, starting with the least-significant bit (that's the bit with weight 2^0).

Then use the table to convert each 4-bit pattern into the corresponding hex digit: "0000" is the hex digit "0", "1101" is the hex digit "D", and "0111" is the hex digit "7".

The resulting hex representation is "7D0".

To prevent any confusion, we'll use a special prefix "0x" to indicate when a number is being shown in hex, so we'd write "0x7D0" as the hex representation for the binary number "0111 1101 0000".

This notation convention is used by many programming languages for entering binary bit strings.