Introduction to Binary Code

Suppose you flip a single fair coin, fairly, twice in a row. What are the possible outcomes? Let's have a think. If H = Heads and T = Tails, then we have:

HH
HT
TH
TT

Now: imagine that you and your friend - who run in the same group - are locked in the same jail cell in the late 19th century Wild West. You've trained for such situations, and have prepared 4 different tactics:

1. lie low and wait,
2. attack the guard,
3. befriend the guard,
4. cause a riot.

Moreover, you've both been clever about how to communicate these tactics. Knowing that when you're in the same cell, the warden can hear anything you say to each other, you've *encoded* these tactics behind the coin flips:

HH = lie low
HT = attack the guard
TH = befriend the guard

TT = cause a riot

Every time you leave camp, you hide a coin in your boot for eventualities like this. Looking at your friend in the cell, you produce the coin slowly. He looks back at you. You flip TH. You're going to try to befriend the guard for now; perhaps something else will come later…!

But suppose you find out through bitter experience that 4 tactics aren't enough: you need more. Suppose you want the following six tactics:

1. lie low and wait,
2. attack the guard,
3. befriend the guard,
4. cause a riot,
5. try digging your way out, à la Shawshank Redemption,
6. pretend to be ill.

Two coin flips are no longer enough to encode all these tactics. We need a third flip of the coin. So how many possible outcomes are there if we flip three coins? Here's the answer:

HHH
HHT
HTH
HTT
THH
THT
TTH
TTT

We can encode our tactics as follows:

HHH = lie low
HHT = attack guard
HTH = befriend the guard
HTT = cause a riot
THH = try digging your way out
THT = pretend to be ill
TTH = undefined

TTT = undefined

Notice that we have two extra codes here that are undefined - we only have six tactics, but we have 2 more possible outcomes of three coin flips.

But now suppose that you and your friend aren't in the same cell anymore; you're in opposite cells. Coin flips will no longer work, as your friend can't see whether Tails or Heads have landed from the opposite cell.

Luckily, you have a backup plan. You've also stashed a mini flashlight in your boot. Suppose 1 means 'ON' (there's a light coming from the flashlight) and 0 means 'OFF', there's no light coming from it. We can also encode our 6 tactics as follows:

111 = lie low
110 = attack guard
101 = befriend the guard
100 = cause a riot
011 = try digging your way out
010 = pretend to be ill
001 = undefined
000 = undefined

Now: both the coin flipping code, and the flashlight code, are ***binary codes***. A binary code is a code comprising symbols that can be in two possible states (H/T, on/off). Binary codes are incredibly powerful and are the basis of computation, electronics, memory, algorithms and thus data science. It's truly fascinating - spellbinding - how such complexity boils down to such simple elements.

Notice how binary code isn't the only way code could be. *Unary* code is code with one symbol (this would be a little pointless)! Ternary code is code with three symbols and didn't have anything like the impact binary code did.

As a general rule, for a binary code of length *n*, there are 2^n possible codes. So where *n*

= 2, we saw there are 2^2 = 4 possible codes, and where *n* = 3, we saw there are 2^3 = 8 possible codes. For *n* = 4, there are 16 codes. If we need to know how long our binary codes need to be in order to encode *k* different things (such as escape tactics) we can find out by getting log2(k) (pronounced 'log base 2 of k); that is, the power to which 2 has to be raised to get k. For example, suppose we need to encode 16 different escape tactics, not 6. So we calculate log2(16). This gives us 4. We thus need binary codes of length 4 to encode 16 different instructions. If we only needed to encode, say, 13 escape tactics, we could still use binary codes of length 4, and just have 3 codes undefined. (Generally, if log2(k) gives us a non-whole number, we find the k needed to get the next-highest whole number, and leave some codes undefined).

In fact, this is how Morse code works. In Morse code, each letter of the English alphabet, and each numeral, is assigned a binary code. Each binary symbol in Morse code is either a dot or a dash. For example, the letter ’S’ corresponds to three dots in a row: ‘. . .’, and the letter ‘O’ corresponds to three dashes in a row ‘_ _ _’. You may have heard the Morse code for ’SOS’ out-loud before: ‘. . . _ _ _ . . .’. Because Morse code encodes letters, rather than whole military tactics, it’s more precise but less fast, than the codes we saw above.