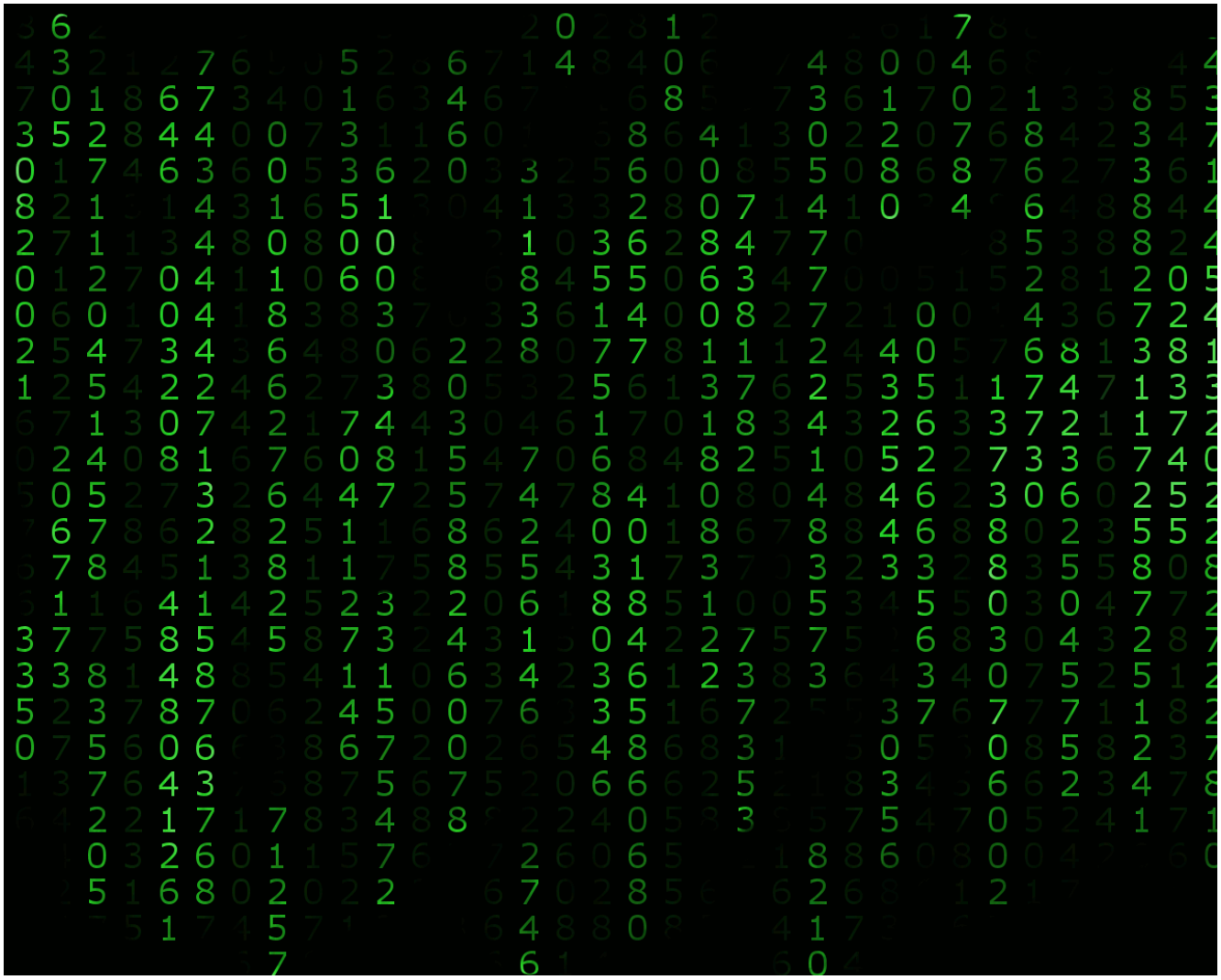
# Numbers

## Count Your Blessings

Techno-Plaza -Oct 1, 2021



# Introduction

Today's kids are taught to approach counting through sets. You count by exclusion. Compare one group to another and note what's not there from the smaller group. All of this is complete abstraction to basic fundamentals of counting. It's like using matrix algebra to balance your checkbook. Oh, I forgot, no one uses checks anymore and reconciled bank accounts are done automatically by a cloud app. The point being; if you want to know how computers work, you need to understand basic counting along with the associated logic circuits that do that counting.

It took mankind millenniums to discover that zero was actually a state that needed to be included in a count. It is possible to have nothing to count but of course, that does matter much when counting products to sell or how many of whatever you get when buying. But as a placeholder for sets of counts turns out to be the key. When the count gets large, you need a way to condense your numbers. It's silly to have a name for each separate count, say past 10, the number of fingers you have. Zero establishes a place holder for sets of counts so you can count up larger numbers easily. Roman numerals eventually got too unwieldily. But it's hard to teach young kids the optimum counting system because it is an infinite power series. Here it is: $...dxB^n + dxB^3 + dxB^2 + dxB^1 + dxB^0 = count$, where the B is the base, i.e., 10, the d is the digit needed to count up to the base value, and the n is the power index that can go to infinity and it all starts with zero. This is not an easy concept to pass on to elementary school kids but the power of it is any counting system can be described by this equation. We are used to base 10 but what's just as important today is base 2 or the binary system where we are only dealing with two digits, "1" and "0", but you only need a "1" to get started. Just keep adding more "1s" to get to any count value...after a lot of tedious counting but computers are very, very fast and don't get bored. You get the same answer as a base 10 ( or any other base) it just takes a lot of steps. There is a gotcha. You have to agree on what $B^0$ means. Ten to the zeroth power is 1 and 0x1 is zero so this first element can be either a 0 or a 1 depending on the digit d, therefore the far right column just mirrors the digits from 0 to 9 with the decimal system or 0 and 1 with the binary system. The binary systems trigger a carry rather quickly, since the right column count can only be 0 or 1, then the next power kicks in at count 2.

The figure on the next page dramatizes this concept and should clear everything up for the skeptics. If you can't accept how binary counting works, then computers will never be your area of expertise. You should move on to painting or sculpture for example.

- Any Number = ...digit x base$^2$ + digit x base$^1$ + digit x base$^0$

$205_{10} = 2 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$ (digits 0 - 9 in decimal)

200 0 5 = 205

$111_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$ (digits 0 or 1 binary)

4 2 1 = 7

$A82_{16} = 10 \times 16^2 + 8 \times 16^1 + 2 \times 16^0$ (digits 0 to F hex)

2560 128 2 = 2690

$YYN_7 = Y \times 7^2 + Y \times 7^1 + N \times 7^0$ (digits yes or no)

49 7 0 = 56

- Gang sign = “dxB$^i$” digit x base(radix) to integer exponent sequence

The reason binary mathematics is chosen as the basis for modern computers is electronic circuits can easily be made to be either switched on or off to match the numbers the user places into the machine so a human to machine interface is realized. Logic gates based on switching can perform all sort of functions from math operations to comparisons but more importantly represent decision points within a digital word of bits or a series of 1s and 0s by abstracting whether an idea is either true or false by assigning that idea to a 1 or 0. Various functions within a computer can be sequenced by binary numbers like memory addresses or the data values at those addresses. The point is whether you're talking about a plain number value that's calculated or the status of the machine, or where in memory to store a value, it can all be abstracted into languages that humans can easily understand without resorting to fumbling with 1s and 0s. It starts with the understanding how the computer is engineered and define various states and operations with a code written in binary. Then various collections of code steps are combined into a usable operations and given a name. The code that represents the machine state of which circuits are on or off determines what the next thing the computer is about to do is called machine code and is a set of 1s and 0s but represents an operation. This operation can be abstracted to a statement a human can read like add for adding two numbers and storing the answer. Then part of the statement includes where the operands are stored and where to place the answer. This level of language is called assembly language. Assembly language statements can then be arranged to perform higher level operations like data structures, memory management and that's where languages like C and C++ were developed. Further advances form instruction shells around C and C++ to capture larger concepts within programs to improve efficiency, routines that perform complex math operations, or data organization like databases. Examples of higher level languages would be JavaScript, Python, Java or any number of frameworks and application specific languages or libraries that encompass modern computing practices. All of it rests on defining the meaning behind a set of 1s and 0s.

The power of a binary basis is you only need a one to start adding up to any other number and decisions are made by determining whether a statement is true or false, which can be abstracted or assigned to 1 or 0. As primitive as this may seem, it is the foundation of all modern computers and how they operate.