BINARY SIMPLIFIED

Exponential Growth Table (Binary)

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1,024
2^{11}	2,048

A single binary digit is referred to as a bit.

A group of four binary digits is a nibble.

A group of eight binary digits is a byte.

Exponential Growth Table (Decimal)

10^0	1
10^1	10
10^2	100
10^3	1,000
10^4	10,000
10^5	100,000
10^6	1,000,000
10^7	10,000,000
10^8	100,000,000
10^9	1,000,000,000
10^{10}	10,000,000,000
10^{11}	100,000,000,000

Anytime you are working with computer data, you are working with bits! Whether it's social media, a document such as this one, on your smartphone, playing a video game, etc. Anytime you interface with an electronic device there are billions upon billions of bits being transferred and/or manipulated at any given time. Hopefully, after reviewing the material I have provided it will give a little insight as well as understanding on how data works and how it is stored.

Bit	Either 0 or 1
Byte	8 bits (10101010)
Kilobyte	1,024 bytes
Megabyte	1,024 kilobytes
Gigabyte	1,024 megabytes
Terabyte	1,024 gigabytes
Petabyte	1,024 terabytes

A bit is the smallest form of data you can have.

A single byte is made up out of 8 bits.

A kilobyte is 1,000 bytes. $1{,}000^0$
A megabyte is 1,000,000 bytes. $1{,}000^1$
A gigabyte is 1,000,000,000 bytes. $1{,}000^2$
A terabyte is 1,000,000,000,000 bytes. $1{,}000^3$
A petabyte is 1,000,000,000,000,000 bytes. $1{,}000^4$

I've underlined the prefixes in each term to show that when working with sizes of data, it is based off exponential increments of 1,000. Starting at 1000^0 which is 1 kilobyte. Anything to the power of 0 will always give a result of 1.

In binary we only work with two digits, 1 and 0.
0 – Off or False
1 – On or True

Imagine we have a light switch. If the switch is in the "On" position, then the light is on. Does the light have power? True. However, if the switch is in the "Down" position, then the light is off. Does the light have power? False. This is an example of a physical switch.

In computers and electronics, we have a similar concept, but instead of physical switches we have electrical switches. In every device that you use there are trillions of tiny components which are known as transistors. Now, transistors have multiple purposes but one of their main uses is to store a value or a bit. Either 0 or 1. On or Off. True or False.

Starting to make a little more sense? Computer logic is nothing more than verified and proven mathematics and real-life concepts that we already use every day.

Here is an example on how a computer stores a word document. Let's say the final size of my document is 20kb or 20 kilobytes.

20 kilobytes * 1,024 = 20,480 bytes.

20,480 bytes * 8 = 163,840 bits.

In the case of a 20kb file I would need a minimum of 163,840 transistors or electrical switches to represent my word document. Pretty crazy right?

Let's figure out how to read and calculate a binary number, also how to convert a binary number between decimal or binary.

First, we must understand another concept. If you remember algebra when you're working with exponents you have a base and a power. The exponent or power stands for how many times the value is being multiplied. The number that is being multiplied is called the base.

Unary – Base 1 (Roman Numerals)

Binary – Base 2

Ternary – Base 3

Octal – Base 8

Decimal – Base 10

Hexadecimal – Base 16

The system that we all know and love, use on an everyday basis is called the decimal system. However, its true mathematical name is Base 10. Why? If you refer to the chart at the top of this document our exponential base is 10 and we increase in increments of 10. The same thing applies to binary, also any other number system in the world. Whatever your base is, is how numbers will be calculated and how you will count. That is what we're going to do today, we are going to re-learn how to count!

Think way back when you first began learning math and learning how to count, or how to write a number, or know what a number means just by looking at it. Say I give you the number 243. Or two-hundred and forty-three. If I asked you how you knew that was 243 what would you say?

We all learned an especially important concept at a young age, the concept of placeholders.

Thousandths = 10^3	Hundredths = 10^2	Tens = 10^1	Ones = 10^0
0	2	4	3

$10^3 * 0 = 0$

$10^2 * 2 = 200$

$10^1 * 4 = 40$

$10^0 * 3 = 3$

$0 + 200 + 40 + 3 = 243$

We know that if we want to represent the number 243 in the decimal system this is how we do so. 0 in the thousandths place, 2 in the hundredths place, 4 in the tens place, and 3 in the ones place. Binary is no different in this manner. The only thing that changes is the placeholders. A number is nothing more than a picture, a symbol, it only holds value because of the placeholder it falls under, otherwise it's just that, a symbol.

Now, let's represent that same number, 243, but we'll represent it in binary this time.

$128 = 2^7$	$64 = 2^6$	$32 = 2^5$	$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
1	1	1	1	0	0	1	1

$2^7 * 1 = 128$

$2^6 * 1 = 64$

$2^5 * 1 = 32$

$2^4 * 1 = 16$

$2^3 * 0 = 0$

$2^2 * 0 = 0$

$2^1 * 1 = 2$

$2^0 * 1 = 1$

$128 + 64 + 32 + 16 + 0 + 0 + 2 + 1 = 243$

243 represented in binary is 11110011. You can represent any number between 0-255 with 8 binary digits or 8 bits. If you wanted to represent a larger number, you just add the next placeholder or a new bit which would be 2^8 or 256 which would then allow you to represent up to 511 and so on and so on. So now that we know how to read and calculate a binary number into decimal let's change a decimal number into binary!

To keep it simple we'll pick an 8-bit number. Say 117. All you must do is take your decimal number and divide it by two continuously until you hit 0. With each division you'll be writing down either a 0 or 1 depending on if you have a remainder. 0 being no, 1 being yes. Also, when you're converting decimal to binary you will be writing it from right to left. This has to do with a concept that is known as MSB and LSB which is your most and least significant bits, again I won't get into specifics, just know for the correct answer you must write it from right to left. Let's finish this example.

I have 117.

117 / 2 = 58.50 (Since I have a remainder, it's 1)

58 / 2 = 29 (No remainder, it's 0)

29 / 2 = 14.5 (Since I have a remainder, it's 1)

14 / 2 = 7 (No remainder, it's 0)

7 / 2 = 3.5 (Since I have a remainder, it's 1)

3 / 2 = 1.5 (Since I have a remainder, it's 1)

1 / 2 = 0.5 (Since I have a remainder, it's 1)

117 in its binary notation is 1110101. Or if you wanted to make it 01110101 for an 8-bit number you can. You can have as many trailing zeros as you want as they hold no value and will not change the number.