introduction to IT support

- Computer Language
- Character Encoding
- Binary
- Supplemental Reading on Logic Gates
- How to Count in Binary

Computer Language

Remember when I said that a computer is a device that
stores and processes data by performing calculations?
Whether you're creating
an artificial intelligence that can be
humans at chess or
something more simple like running a video game,
the more computing power you have access to,
the more you can accomplish.
By the end of this lesson,
you'll understand what a computer calculates and how.
Let's look at this simple math problem.
Zero plus one equals what?
It only takes a moment to come up with the answer one,
but imagine that you needed to do
100 calculations that were this simple.
You could do it, and if you are
careful you might not make any mistakes.
What if you needed to do 1,000 of these calculations?
How about a million?
How about a billion? This is
exactly what a computer does.
A computer simply compares ones and zeros,
but millions or billions of times per second.
[inaudible]. The communication that a computer uses is
referred to as binary system,
also known as base-2 numeral system.
This means that it only talks in ones and zeros.
You may be thinking,
my computer only talks in ones and zeros.
How do I communicate with it? Think of it like this.
We use the letters of the alphabet to
form words and we give those words meaning.
We use them to create sentences,
paragraphs and whole stories.
The same thing applies to binary,
except instead of A, B, C,
and so on, we only have zero
and one to create words that we give meaning to.
In computing terms, we group
binary into eight numbers or bits.
Technically, a bit is a binary digit.

Historically, we use
eight bits because in the early days of computing,
hardware utilized the base-2 numeral system
to move bits around.
Two to the eighth numbers offered us
a large enough range of
values to do the computing we needed.
Back then, any number of bits was used,
but eventually the grouping of eight bits
became the industry standard that we use today.
You should know that a group of eight bits
is referred to as a byte.
A byte of zeros and ones could look like 10011011.
Each byte can store one character,
and we can have 256 possible values
thanks that a base-2 system, two to the eighth.
In computer talk, this byte
can mean something like the letter
c. This is how a computer language is born.
Let's make a quick table to translate something
a computer might see
into something we'd be able to recognize.
What does the following translate to?
Did you get hello? Pretty cool.
By using binary, we can have
unlimited communication with our computer.
Everything you see on your computer right now,
whether it's a video,
an image, texts or anything else,
is nothing more than a one or a zero.
It is important that you understand how binary works.
It is the basis for everything
else we'll do in this course.
Make sure you understand the concept before moving on.

Character Encoding

By the end of this video,
you'll learn how we can represent the words, numbers,
emojis, and more we see on our screens from
only these 256 possible values.
It's all thanks to character encoding.
Character encoding is used to assign
our binary values to
characters so that we as humans can read them.
We definitely wouldn't want to see
all the texts in our emails in
webpages rendered in complex sequences of zeros and ones.
This is where character encodings come in handy.
You can think of character encoding as a dictionary.
It's a way for your computers to look up
which human character should be
represented by a given binary value.
The oldest character encoding standard used is ASCII.
It represents the English alphabet,
digits, and punctuation marks.
The first character in the ASCII to
binary table, a lowercase a,
maps to 01100001 in binary.
This is done for all the characters
you can find in the English alphabet,
as well as numbers and some special symbols.
The great thing with ASCII was that we only needed to use
127 values out of our possible 256.
It lasted for a very long time,
but eventually, it wasn't enough.
Other character encoding standards were
created to represent different languages,
different amounts of characters, and more.
Eventually, they would require
more than 256 values we are allowed to have.
Then came UTF-8,
the most prevalent encoding standard used today.
Along with having the same ASCII table,
it also lets us use a variable number of bytes.
What do I mean by that? Think of any emoji.

It's not possible to make emojis with
a single byte since we can
only store one character in a byte.
Instead, UTF-8 allows us to
store a character in more than one byte,
which means endless emoji fun.
UTF-8 is built off the Unicode Standard.
We won't go into much detail,
but the Unicode Standard helps us
represent character encoding in a consistent manner.
Now that we've been able to represent letters, numbers,
punctuation marks, and even emojis,
how do we represent color?
Well, there are all kinds of color models.
For now, let's stick to a basic one
that's used in a lot of computers,
RGB or red, green, and blue model.
Just like the actual colors,
if you mix a combination of any of these,
you'll be able to get the full range of colors.
In computer learn,
we use three characters for the RGB model.
Each character represents a shade of the color,
and that then changes the color
of the pixel you see on your screen.
With just eight combinations of zeros and ones,
we're able to represent everything
that you see on your computer from
a simple letter a to the very video
that you're watching right now. Very cool.

Binary

You might be wondering how are computers get these ones and zeros?
It's a pretty question. Imagine we have a light bulb and a switch that turns the state of the light on or off.
If we turn the light on, we can denote that state is one, if the light bulb is off, we can represent the state as zero.
Now imagine eight light bulbs and switches that represents eight bits with a state of zero or one.
Let's backtrack to the punch cards that were used in Jacquard's loom.
Remember that the loom use cards with holes in them.
When the loom would reach a hole, it would hook to thread underneath, meaning that the loom was on.
If there wasn't a hole, it would not hook the thread, so it was off.
This is a foundational binary concept. By utilizing the two states of on or off, Jacquard was able to weave
intricate patterns into fabric with his looms.
Then the industry started refining the punch-cards a little more.
Where there was a whole, the computer would read one, if there wasn't a hole, it would read zero.
Then by just translating the combination of zeros and ones, a computer could calculate any possible amount of numbers.
Binary in today's computer isn't done by reading holes.
It uses electricity via transistors allowing electrical signals to pass through.
If there's an electric voltage, we would denote it as one, if there isn't, we would denote it by zero.
But just having transistors isn't enough for our computer to be able to do complex tasks.
Imagine if you had two light switches opposite ends of a room, each controlling of light in the room.
What if when you went to turn on the light with one switch, the other switch wouldn't turn off?
That'll be a very poorly designed room.
Both switches should either turn the light on or off, depending on the state of the light.
Fortunately, we have something known as logic gates.
Logic gates allow our transistors to do more complex tasks like decide where to send electrical signals
depending on logical conditions.
There are lots of different types of logic gates, but we won't discuss them in detail here.
If you're curious about the role that transistors and logic gates play in modern circuitry, you can read more about it in the supplementary reading.
Now we know how our computer gets it's ones and zeros to calculate into meaningful instructions.
Later in this course, we're going to be able to talk about how we're able to turn human-readable instructions into zeros
and ones that our computer understands through compilers.
That's one of the very basic building blocks of programming that's led to the creation of our favorite social media sites, video games, and just about everything else.

Supplemental Reading on Logic Gates

Logic Gates

Knowing how logic gates work is important to understanding how a computer works. Computers work by performing binary calculations. **Logic gates** are electrical components that tell a computer how to perform binary calculations. They specify rules for how to produce an electrical output based on one or more electrical inputs. Computers use these electrical signals to represent two binary states: either an “on” state or an “off” state. A logic gate takes in one or more of these binary states and determines whether to pass along an “on” or “off” signal.

Several logic gates have been developed to represent different rules for producing a binary output. This reading covers six of the most common logic gates.

Six common logic gates

NOT gate

The NOT gate is the simplest because it has only one input signal. The NOT gate takes that input signal and outputs a signal with the opposite binary state. If the input signal is “on,” a NOT gate outputs an “off” signal. If the input signal is “off,” a NOT gate outputs an “on” signal. All the logic gates can be defined using a schematic diagram and truth table. Here’s how this logic rule is often represented:

Not gate schema and truth table

On the left, you have a schematic diagram of a NOT gate. Schematic drawings usually represent a physical NOT gate as a triangle with a small circle on the output side of the gate. To the right of the schematic diagram, you also have a “truth table” that tells you the output value for each of the two possible input values.

AND gate

The AND gate involves two input signals rather than just one. Having two input signals means there will be four possible combinations of input values. The AND rule outputs an “on” signal only when both the inputs are “on.” Otherwise, the output signal will be “off.”

AND gate schema and truth table

OR gate

The OR gate involves two input signals. The OR rule outputs an “off” signal only when both the inputs are “off.” Otherwise, the output signal will be “on.”

OR gate schema and truth table

XOR Gate

The XOR gate also involves two input signals. The XOR rule outputs an “on” signal when *only one* (but *not both*) of the inputs are “on.” Otherwise, the output signal will be “off.”

XOR gate schema and truth table

The truth tables for XOR and OR gates are very similar. The only difference is that the XOR gate outputs an “off” when both inputs are “on” while the OR outputs an “on.” Sometimes you may hear the XOR gate referred to as an “exclusive OR” gate.

NAND gate

The NAND gate involves two input signals. The NAND rule outputs an “off” signal only when both the inputs are “on.” Otherwise, the output signal will be “on.”

NAND gate schema and truth table

If you compare the truth tables for the NAND and AND gates, you may notice that the NAND outputs are the opposite of the AND outputs. This is because the NAND rule is just a combination of the AND and NOT rules: it takes the AND output and runs it through the NOT rule! For this reason, you might hear the NAND referred to as a “not-AND” gate.

XNOR gate

Finally, consider the XNOR gate. It also involves two input signals. The XNOR rule outputs an “on” signal only when both the inputs are the same (both “On” or both “Off”). Otherwise, the output signal will be “off.”

XNOR gate schema and truth table

The XNOR rule is another combination of two earlier rules: it takes the XOR output and runs it through the NOT rule. For this reason, you might hear the XNOR referred to as a “not-XOR” gate.

Combining gates (building circuits)

Logic gates are physical electronic components—a person can buy them and plug them into a circuit board. Logic gates can be linked together to create complex electrical systems (circuits) that perform complicated binary calculations. You link gates together by letting the output from one gate serve as an input for another gate or by using the same inputs for multiple gates. Computers are this kind of complex electrical system.

Here’s a schematic drawing for a small circuit built with gates described above:

Combined circuit schematic

Here is the truth table for this circuit:

Combined circuit truth table

This circuit uses three logic gates: an XOR gate, a NOT gate, and an AND gate. It takes two inputs (A and B) and produces two outputs (1 and 2). A and B are the inputs for the XOR gate. The output of that gate became the input of the NOT gate. Then, the output of the NOT gate became an input for the AND gate (with input A as the other). Output 1 is the output from the AND gate. Output 2 is the output from the XOR gate.

Key takeaways

Logic gates are the physical components that allow computers to make binary calculations.

Logic gates represent different rules for taking one or more binary inputs and outputting a specific binary value (“on” or “off”).

Logic gates can be linked so that the output of one gate serves as the input for other gates.

Circuits are complex electrical systems built by linking logic gates together. Computers are this kind of complex electrical system.

How to Count in Binary

Binary is the fundamental communication block of computers, but it's used to represent more than just text and images. It's used in many aspects of computing, like computer networking, what you'll learn about in a later course. It's important that you understand how computers count in binary. We've shown you simple look up tables that you can use like the ASCII binary table. But as an IT support specialist whether you're working on networking or security, you'll need to know how binary works, so let's get started. You'll probably need a trusty pen and paper, a calculator and some good old fashioned brainpower to help you in this video. The binary system is how our computers count using 1s and 0s, but humans don't count like that. When you were a child you may have counted using ten fingers on your hand, that innate counting system is called the decimal form or base ten system. In the decimal system there are ten possible numbers you can use ranging from 0 to 9. When we count binary which only uses 0 and 1, we convert it to a system that we can understand, decimal. 330, 250 to 44 million, they're all decimal numbers. We use the decimal system to help us figure out what bits our computer can use. We can represent any number in existence just by using bits. That's right, we can represent this number just using ones and zeros, so how does that work? Let's consider these numbers, 128, 64, 32, 16, 8, 4, 2 and 1, what patterns do you see? Hopefully you'll see that each number is a double of the previous number going right to left, what happens if you add them all up? You get 255, that's kind of weird, I thought we could have 256 values for a byte. Well, we do, the 0 is counted as a value, so the maximum decibel number you can have is 255. What do you think the number is represented here? See where the 1s and the 0s are represented? Remember, if our computers use the 1 then the value was on, if it sees a 0 then the value was off. If you add these numbers up you'll get a decimal value. If you guess 10, then you're right, good job, if you didn't get it, that's okay too, take another look. The 2 and 8 are on and if we add them up we get 10. Let's look at our ASCII binary table again, the letter h in binary is 01101000. Now let's look at an ASCII to decimal table. The letter h and decimal is 104. Now let's try our conversion chart again, 64+32+8=104. Look at that, the math checks out. Now we're cooking, wow, we've gone over all the essentials of the basic building blocks of computing and machine language.