DOUG LLOYD: If you've been watching these videos in the order which we recommend, we're about to undergo bit of a culture shift. Because now, we're going to start talking about the internet and web technologies. So up until now, we've really been doing a lot of C. And when we've been running our programs, we have been running them from the command line. That's pretty much how the users have been interacting with the programs that we write. They pick something to prompt, something happens in the terminal window, and then it's done.

Sometimes you might have persistent data that remains afterwards. But that's pretty much it. It's at the command line. It's the only way the user can interact. From this point forward, we're going to start transitioning so that the users can interact with our websites. So we're going to be writing websites, which aren't written in C, but are written in a variety of other programming languages, including PHP, and it's sort of helper languages, HTML, CSS, and the like. So we're going to start talking about those things.

Before we get into web programming itself, I think it's probably a good idea to take a step back and talk about how computers and humans interact over the web. So this video is really a primer, a basic guide, to the internet. Now, the caveat here is the CS50 is not a networking class. So what we're going to be talking about here is pretty high level. We're not going to get into any low level details of how all this stuff works. If you're interested in that, I'd strongly recommend taking a class on computer networking. And we might even tell white lie or two just for the purposes of making the general understanding clear.

So with that said, let's talk about how we interact with the internet. So here we are. Here's us. We're pretty looking forward to getting onto the internet, which as we all know, is chock full of cats.

Now do we just connect to the internet like this? Well, probably not. Intuitively, you know that, say for example, when you change your Wi-Fi network on your computer, you don't see one called internet unless that just so happens to be the name of your local Wi-Fi. Right?

It's usually something like home. Or if you're at work, it might be the name of your company. There's not just one option called internet. And so something or some things exist in between when we want to connect to the internet. What are some of those things? Well, we're going to talk about that. We're also going to talk about some of the important things we need in order to be able to connect to the internet. And the first of these things is an IP address. So you've probably heard the term IP address before. What does it mean? Well, an IP address is basically a unique identifier of your computer on a network. Just like every home or office has a unique address to which one could send a mail.

Similarly, every computer if it wants to receive data or send data, needs to have a unique address. So that when information is sent or received, it's being sent from or received to the correct location. This addressing scheme, as I said, is called IP addressing. IP is stands for Internet Protocol, which we'll talk about again shortly.

Now, what does IP addressing look like? Well, the scheme basically was, when it was first implemented, to give every computer a unique 32-bit address. That's a lot of bits. That's 4 billion addresses.

And generally, instead of using hexadecimal notation, which we've used previously in the context of pointers in C to talk about addresses, we usually represent IP addresses in a little bit more of a human friendly way, representing them as four clusters of 8 bits represented as decimal numbers. Because humans don't frequently speak hexadecimal, unless you're programming. But people who use the internet aren't necessarily programmers.

And so making it easy and accessible for them to be able to talk about what their IP address is in case they maybe need to call up somebody to troubleshoot something, it's better to make it in the more common conventional decimal number format. And so an IP address just looks pretty much like this, w.x.y.z, where each one of those letters represents a non-negative value in the range of 0 to 255. Recall that an 8-bit number can hold 256 distinct values.

And so that's why our range is 0 to 255. And we have four clusters of 8 bits for a grand total of 32 bits. And so an IP address might look something like this. This is sort of a generic default IP address, 123.45.67.89. All of them are in the range of 0 to 255, so that's a valid IP address.

Here at Harvard University, all of our IP addresses start with 140.247. That's just the way that the IP addresses in this geographic area have been assigned. And so this might be an IP address that might exist here at Harvard.

So as I said, if every IP address is 32 bits, we have about 4 billion to give out, a little more than 4 billion. But we can kind of see a problem, right? What's the world population right now?

Well, it's somewhere north of 7 billion people. And in the Western world at least, most people have more than one device capable of internet connectivity. I have one right here. And I have another one in my pocket. And I have one back in my office.

And so that's three. And that doesn't even count the ones that I have at home, too. And so that's kind of a problem, right? We have at least 7 billion people and only 4 billion addresses.

And every device is supposed to be uniquely identified. We have developed some workarounds to deal with this problem, something called a private IP address, which we're not going to get into in this video. But basically, it allows further the web, the internet, to kind of fake out a little bit that you have a unique address by having private addresses and then funneling them through one single address, which is shared by many different computers.

But that's really not a long term fix. Even that fixed isn't going to last forever. And so we need to have a different way of dealing with this.

So as I said, we had about 4 billion. But that's not going to be good enough, right? And so the way that it has been decided there we're going to deal with this is to make longer IP addresses. Instead of 32-bit addresses, we're going to have 128-bit addresses. So instead of 4 billion addresses, we're going to have that huge number of addresses, which is 340 billion billion billion billion, so a lot of IP addresses.

And this new scheme is called IPv6 is commonly how it's referred. The old scheme being IPv4. It's a bit of a problem in that this problem has been known about for a really long time.

And you'll see this a lot in the context of computers and computing. We're good at anticipating problems. But we're bad at dealing with them even though we know about them. So IPv6 has been around for a while. And only in the last couple years have we actually started phasing in these IPv6 addresses to phase out the IPv4 addresses. But some places do have them. And they look similar to a regular IP address. But they are a lot longer.

So instead of now having four clusters of 8 bytes for your address, we now have eight clusters of 16 bytes. And 8 times 16 is 128. And we represent these in the less conventional hexadecimal form. Because having 16-bit numbers means that instead of being a range of 0 to 255, We'd have a range of 0 to 65,535.

And so having a bunch of those stuck together would be very difficult to read. And so we usually use hex just out of convenience. And so a typical IPv6 address might look something like this.

It's certainly a lot longer than the IPv4 address we've seen before. But this would be a valid IPv6 address. This one is also about IPv6 address.

This one happens to belong to Google. And notice there's a bunch of zeros there. Sometimes these addresses can get so long. And since we're still pretty early in IPv6, sometimes there can be big chunks of zeros in there that we don't need.

If you're reading this out loud, it's 2001.4860.4860.0.0.0.0.8844. It's kind of a lot, right? So if you see a bunch of zeros, you might sometimes see an IPv6 address like this, where they omit the zeros and use a double colon instead. This is OK, though. Because we know that there are supposed to be eight distinct chunks. And so by implication, we see four. So we know that there must be four sets of zeros like this, that fill it in.

So sometimes, you might see an IPv6 address not having eight separated chunks like we do here. You might see it looking like this. And that just means that everything you don't see in between where that double colon is is just zero separated.

So, OK. We know a little bit more about IP addresses now. But how do we get them? We can't just pick the one we want. If we did that, we might end up fighting somebody for the same IP address. Or somebody might have chosen it previously. If we try and take it, we're going to run into a bit of a problem. And so we can't just pick the IP address that we want.

So the way that we get an IP address is somewhere between our computer and the internet, that big internet out there, there's something called a DHCP server, a Dynamic Host Configuration Protocol server. It's a big mouthful of text. But really all it does is it assigns you an IP address.

Your DHCP server has a list of addresses that it can validly assign. And it gives you one. That's pretty much all there is to it. Now before DHCP, this task of assigning addresses fell to a system administrator. So an actual person would have to manually assign your computer and address when you connected to a network. So DHCP just sort of automates this process of giving you an IP address. But that's how you get it. It's just a program running somewhere between you and the internet that has a bank of IP addresses that it can give out. And when you connect to the network, it gives you one. So let's revisit this diagram. Somewhere between you and the internet, there's a DHCP server. OK. So that's good. Now, let's talk about DNS. So we've talked although these IP addresses. And we know that if we're going to uniquely identify a device on the internet, it has to have a unique address.

And we could visit that address if we wanted to. But you've probably never typed in something like 192.168.1.0 into your browser, right? You don't type in numbers into your browser. You usually type in human readable names like google.com or cs50.harvard.edu, right?

Those aren't IP addresses, though. So exists this service called the Domain Name System, DNS, that translates IP addresses to human comprehensible words or phrases that are much more memorable than remembering a set of four numbers or, soon, a set of eight hexadecimal numbers. That would be really challenging, right?

Think about before the days of cell phones. You had your memorize your friend's phone numbers. It might have gotten tough after a little while. And similarly, if you want to visit a bunch of websites, you probably don't want to remember a bunch of numbers. You'd rather remember a bunch of words.

So this mapping, this translating, of sets of numbers to human readable names kind of makes DNS the yellow pages of the web. And you can think about it as if it's just a huge list running from 0.0.0.0 all the way down to 255.255.255.255, which would be the highest possible-- that's the full range from 0s to 255s of all 4 billion-ish IPv4 addresses. I made up the ones on the top and the bottom. But the one in the middle there is actually an IP address. So if we visited 74.125.202.138, apparently that translates to that site there, io-- what the heck is that? Well, not every name that maps is actually clear what it is, right?

So sometimes somebody who owns an IP address might name their host something that they're actually not. For example, that IP address if you went there, is actually just google.com. But Google has a lot of different servers.

And they can't call them all google.com. So they have their own internal system for translating google.com to whatever server actually is connected to that IP address. And then there's another system that exists between to translate that gobbledygook here to google.com. But we won't get into that.

And similarly for IPv6s, we're also going to have a yellow pages that'll be a lot bigger. And similarly, in the middle there-- it was tough to find an IPv6 address that was legitimate. But I found one for Google.

But it's Google's Irish website. But if you went to that IPv6 address, if your browser was IPv6 capable, that would bring you to Google's Irish homepage. So there you go.

But this isn't entirely true, right? This the system seems cumbersome, right? If there's a huge list of 4 billion things to have to look up, that's pretty big. There's no yellow pages of the world, right? If you still get the yellow pages delivered to you-- I got mine the other day, and I just recycled it. But if you do get the yellow pages delivered to you, you don't get a book that's every phone number that exists on the planet, right? You get a list of the local phone numbers, the ones you're most likely to call.

And that's actually what DNS is. If you think about it, DNS is really the local yellow pages. And large DNS servers like google.coms, they are actually just more like libraries that have a copy of all of the local yellow pages or all of the local DNS records. So there's really no one repository of the full DNS of the internet, just like there's no one yellow pages of the world.

There are all these local small scale DNSs that exist out there. And there are services that aggregate them together. But they depend on those smaller DNS systems updating their information, so that they have the most accurate information.

So again, this analogy is large aggregating DNS systems are like libraries that have a copy of every yellow pages of the world. They don't themselves update those books. They depend on the books coming in, so they can update the information if they need it.

So the DNS system is not a giant block. It's decentralized across many, many servers. So now we know that somewhere between us and the internet there exists a DNS server as well as a DHCP server.

Now, access points, what our access points? Well, access points you're probably pretty familiar with from actually connecting to the internet. That's the network that you choose, the home or your work network or what have you.

And I'm generalizing the concept of an access point here for purposes of this video. But there are actually a lot of things that can be rolled up into access points. There are concepts of routers, which is sort of a general term that we use.

But there are also switches and things actually called access points that are separate from this general concept of an access point. But basically what happens is with IPv4, I said we have this concept of private addresses, right? And instead of every machine having a unique IP address, which we have run out of, because we're over 4 billion devices trying to connect to the internet, what we do is instead assign an IP address to a router. That router or access point just in your home, for example.

And the router's job as to sort of act as a traffic cop, allowing everybody who's connected to that router to use the same IP address to get out. Does that make sense? So everybody at your home has a private IP address. They can't connect to the internet, or the internet rather can't speak to them, through that private address. They can only speak to them through the address in the router. And it's the router's job to take information that you're sending the router and direct it to the correct place and for information that's coming into the router for the router to send it to you.

So the routers are really the devices here-- particularly a router in your home, the most common sort of usage case for most people-- that has the public IP address. That's the device that's connected to the internet. And you connect to the router to have information flow through it on your behalf.

As I said, a modern home network, the router and switch and access point are all kind of bundled up into a single device. Sometimes a modem is bundled in there as well. That's usually just called a router. But it's really all of those things together.

Large scale business networks or so-called Wide Area Networks, WANS, actually keep these devices separate. They have a switch. They have routers. They have multiple access points.

For example, at a university you'll see things that look like so-called routers mounted are all around campus. Those are all access points that flow into routers, switches, et cetera, to pass information along. Because these networks are so big that one single access point can't cover its large area.

And so these large networks, business networks, et cetera, split these into separate devices, so the network and scale and grow if needed. So again, somewhere between us and the internet, we have an access point. And that's what we connect to. And through there, we can get to the internet. As I said at the beginning of this video, this is not a course on networking. So this is not the entire story. And I've kind of glossed over it. And maybe I've left you even a little bit confused as to what some of these things are. But that's OK.

We don't need the whole story. It's enough for us to know moving forward just basically a little bit about how the internet works. So what we know is we have these private networks at our house.

And we connect to a router. And that router is connected to the internet at large. But what is the internet at large? I keep saying this, but what is it?

Well, it's really just all these individual networks at my house, and at your house, and at every other house, that are connected together. It's an interconnected network, an inter-net. So instead of thinking about the internet as this giant cloud, this ethereal thing that exists out there, it's really just a connection among all of these networks.

So here we go. We have our local network. And we're not the only person probably on our local network trying to use the internet. There's probably several of us trying to get in.

And we're not the only network that exists in the world, right? There are other networks, too, that are trying to connect to the internet. But the internet is not, again, a separate entity.

It's just a set of rules that allow these networks, these small networks, the blue, the purple, and the red network here, to communicate with each other. So there's no thing they're all connecting to. They're all just connected to each other, right?

And so somewhere on these networks exists the services that we actually want. So maybe in the blue network is where Google lives. And in the purple network is where Facebook lives. And in the red network, well, maybe that's where all those cats are.

And so if we want to get information about cats, we just traverse this chain of networks to get the information we want. And here, I've represented the network as all being able to talk to each other. And we can only talk to the network. But the network can't talk back to us.

But that's not true either, right? This is all a two-way street. Information can flow through networks back and forth.

How does it do that? Well, the internet's really a system of protocols. And we're going to start talking about what those protocols are in future videos.

But again, the internet is not a separate thing. It's a set of rules that defines how networks communicate, these small networks, these local network that we're used to, the people in our house, the people at our school, the people at our job, all sharing a network. And how these networks interconnect and talk to each other, that's actually what the internet's all about. So let's, in a future video, talk about some of the protocols that comprise the internet to hopefully give you a bit more of a well-rounded understanding. I'm Doug Lloyd. This is CS50.

**IP**

DOUG LLOYD: If you watched our internet primer video, I left a bit of a cliffhanger by talking about the internet and how it's a system of protocols. Well, let's talk about the first of those protocols that actually comprises the internet. And interestingly enough, it's called the Internet Protocol, which we usually refer to as IP. So the internet, as I said, is an interconnected network, an internet, which is really just several networks woven together and agreeing somehow to communicate with one another. What is this somehow I'm talking about? Well, this is the Internet Protocol. This dictates how information is transmitted from point A to point B. And this is sort of a condition of joining the network of the internet is agreeing to follow this protocol when information needs to be moved from point A to point B.

So at the very end of that internet primer video, I showed this image of what the internet was. And on a small scale, this is actually probably pretty accurate. This might be how three networks actually talk to each other. But it's a bit misleading. And the reason it's a bit misleading is because-- if I just number the networks for the sake of convenience here and we get rid of everything else and just focus on the networks-- it's a bit misleading because it implies that all three network have a connection to one another.

One is connected to two. Two is connected to three. And three is connected to one. And when I talk about a connection here, I'm talking about a physical, wired connection. We do have wireless. But it's really impractical for data to be transmitted wirelessly over a large scale. And so at some point, we really do rely on wired technology-- telephone wires, fiber optic wires, various technologies that are physically connecting point A to point B.

And on a small scale like this-- this might be accurate, but as the image gets a little bigger, let's now imagine we have six different networks. If that's true, now we have something like this for every network to be connected to every other network. And if you look, every network has five arrows connected to it. So everything is connected to every other network.

We only have six networks here, and already look at how much wiring we have to employ, right? And the internet consists of a lot more than six networks. We can't afford to wire each network to each other network, especially considering some of these networks span oceans, right? If we're trying to connect to a network in Asia or in Europe, we're going to have to span an entire ocean. We're going to need to use wires at some point, but we want to minimize the number of wires we actually use. We don't want to send a million wires across the ocean, because they cost millions of dollars apiece to lie down. And so rapidly, we wouldn't be able to afford the internet anymore. So we have to have another way for every network to talk to every other network or else we have pieces of the internet that are disconnected from other pieces of the internet. And that's not what we want. But we don't want to have them all wired together.

And this is where routers come back into play. We can use routers in the following way. What if instead of every network being physically connected to every other network, we had these intermediary pieces, where the networks were connected to these intermediaries, which are connected to a few networks. So instead of having one connect to two, three, four, five, six, maybe one connects to a router, which maybe connects to one or two of those networks, but also maybe connects to other routers, which also will connect to those other networks.

And the router's job is-- it contains information called a routing table that dictates where do I go if I see a particular IP address? If I see an IP address starting with four, I'm going to go this way. If I seen IP address starting with a 12, I'm going to go that way. We don't need to be connected physically to network number four or network number 12 in this example. We just know generally where we want to go. And if you think about it, this is sort of similar to the concept of recursion that we talked about when we were talking about it in C. I'm not going to connect you to exactly where you want to go. I'm just going to move you one step closer to where you want to go. And I'll let somebody else deal with solving the rest of the problem. I'll just solve this little piece of the problem and defer the rest of it to somebody else. So routing information is actually kind of similar to recursion. If that's a concept that you understand well, maybe that analogy would help.

So let's take a look at this networking example again and assume that, again, we're going to use those same six networks, one through six. So let's just say that every IP address on network one starts with one dot something. And we'll say that there's some other thing that deals with how all the systems are connected to network one. We just care about connecting all of those networks together in an internet. So every device that is connected to network one has an IP address that starts with one dot and then three other numbers.

This is a generalization of the way things actually work. It's quite a bit more precise than this. But this should give you a general idea of what the Internet Protocol is actually doing. So this was the diagram we had before. This was the system that was not sustainable. Even six, this might be OK. But if we get to 10 or 20 or 50, we're going to be lying a lot of wires. And 50 is still also not even the tip of the iceberg as to the number of networks we have. So this model is unsustainable. We can't stick with this.

So let's instead adopt this model where we get rid of all the wires between the networks and we add routers. So these yellow boxes represent routers. And their job is to move information generally closer to where it's supposed to go. And maybe these are the connections that these networks have. And maybe these are the tables that are built into the routers.

So if we just start by looking at network one, for example, basically what it says is if I ever see an address that doesn't start with a one-- that's what the exclamation point one or the bang one there, not one-- I'm going to pass it off to a router. And from there, the router can make a decision. The router says if I see a one, I'm going to move to network number one. That's the green arrow heading to the left out of that top left box.

If I see a two-- that's the arrow sort of heading to the top right towards the purple network-- if I see an IP address starting with a two, I'm going to go towards the two network. If I see a three, a four, a five, or a six-- that's that red arrow coming out of the top left router-- I'm not connected to three, four, five, or six. But I know somebody who is or who's a little bit closer to there. So I'm just going to say, every time I see an IP address starting with three, four, five, or six, I'm just going to send it to that router. So I'll move it a little closer to where it's supposed to go and let that router deal with the problem.

And as you can see-- if you wanted to pause here and trace-- you can get to every other point in the network from wherever you are. All six networks can still connect to every other network but they're not physically connected anymore. They're now these intermediate steps. Now, of course there's a trade off of speed, right? If one was directly connected to six, we wouldn't have to go through two routers along the way. So we may be able to get the connection a little bit faster. But maybe that trade-off is worth it, right? If it's going to be so expensive in terms of actual cost, dollars and cents, to physically wire all these networks together, maybe a little bit of a slowdown in speed is OK. We can tolerate that.

So again, in that example we were just talking about, none of the networks directly connect to each other all. There could have been-- maybe in that example we could have made it so that maybe network one and two were directly connected. And that would be OK. Some networks are physically connected to other networks. But they're not all connected to each other. They rely on the routers-- in this particular example-- to distribute the communication from point A to point B. On a small scale-- like what we're talking about here-- this configuration actually might be more inefficient than just having direct connections. But on a large scale, we can scale the system a lot better. It's really going to reduce our cost of network infrastructure to have intermediary routers whose job it is to move traffic from the sender to the receiver, from point A to point B, as opposed to wiring everybody together.

So let's take a look at an example of information traveling using this Internet Protocol. Let's say that I am physically located at IP 1.208.12.37 so I exist somewhere on the one network. And I want to send a message to you. And you're on the five network at 5.188.109.14. Your IP address specifically doesn't matter, but in this particular example we're talking about this generalization of what the internet protocol is all about. You're on the five network, and I'm on the one network. As you can see, we're not connected to each other at all.

So I start out. And I want to send you a message. And so somehow I communicate that message to the router. The router is the one that actually has the IP address. And it's looking at where it's supposed to go. We're going to five dot something. So now I'm going to start using my-- or the router, rather, is going to start using its router table to pass information along. It sees that five is not one, so it says I'm going to pass it to this guy. Then this guy has to make a decision. Where am I going to go? Well, it's not a one, so I'm not going to move to the one network. And it's not a two. I'm not going to move to the two network. It starts with a five. I'm not connected to five, this router says. And so I'm just going to pass it off to-- I'm going to go down this path. This is where threes and fours and fives and sixes go. And I'll let that guy deal with it. I'll get it a little closer to where it's supposed to go. I know it's supposed to go in that general direction. But maybe that guy can deal with it. OK. So that guy looks. He says, OK, this IP address starts with a five. Well, I'm connected to three and to six, so I can't get the message directly where it needs to go. But that other router over there, I know if I send it fours and fives, it can handle those.

So it passes it along down the path. And then this router says, well, I'm connected to networks four and five. So, yes, I can help you. I'll take your IP address that starts with a five. I'll give it to the five network. The five network will do some work on its end and give the message to you. And now we've successfully transmitted a message from me to you using the Internet Protocol.

Again, very generalized for purposes of illustration as to what's happening. But that's pretty much how the Internet Protocol works. The routers know generally where to send it and will send it one step along the way, getting it closer and closer to its destination until one router is physically connected to the network or the address or whatever in question and gives it there.

Now, in general, except for really, really small, small messages, it's not going to send it as one big chunk of data. If I'm sending you an email-- a very long email, say-- it's not going to take that entire email, bundle it up in a ball or a package or whatever, and send that entire thing down the network.

First of all, sending information along the network is expensive. It does add up. And the larger the chunk, the more costly it is to move every step of the way. And if there's somehow a slowdown and then there's this giant-- sort of like if you're driving on the highway and there's this giant truck kind of blocking the way and you can't get around it on either lane because it's kind of spread out. It slows everybody else down behind it. But small cars, if they were all small cars, they might be able to move around, if that analogy sort of helps a little bit.

So one big block in the system can really slow everybody else down. And so what IP is going to do is split this data into packets. It's going to take this big email or FTP transfer or a file transfer, or maybe I'm making a request to a web browser because I want a picture of cat. And it's going to take that request or that email or that file and break it up into many pieces and send all of the pieces separately. So in fact, I'm filling the highway with a lot of small cars, which can all move instead of a big truck that might, if something goes wrong, throttle the traffic for everybody else.

Another side effect of this is if there's some sort of catastrophic failure and something goes wrong and the packet gets dropped. Something is failed and the message can't be communicated. The router maybe had too much stuff going in. It couldn't juggle everything. And so it just literally dropped it. That's sort of the analogy, right?

It's got a lot of things going on. It's passing information from point A to point B. We're not the only two people on the internet, so it has to process a lot of traffic. And if it doesn't have enough hands and it can't figure out what it's doing, it might just drop something. So it can do something else. It's got too much going on.

If we had our message as one huge block and that was what got dropped, now we have to send the message again. And we are now possibly causing traffic again. And we run the risk of that huge block being dropped again. But if the data's been broken up into packets and we drop one of those, it's a lot less costly to send that packet one more time as opposed to the entire thing one more time. So IP is responsible for getting information from point A to point B and also breaking the information into small pieces so that the network isn't overly taxed.

IP is also known as a connectionless protocol. There's not necessarily a defined path from the sender to the receiver or vice versa. Now, in this example we've talked about, there actually is only one way to get to every network. So in this particular illustration, there actually is a defined path from point A to point B. But we can change that by just making one modification to the two routers on the left by adding this condition to the router tables.

Now notice that from the top left router, there are actually two ways to deal with a four or a five IP address. It can go down to the lower left router, or can go to the right, to the right router. It has multiple options. And this is actually kind of a good thing because it makes our network more responsive.

If for example-- it's sort of like a GPS. If you've ever been driving on the highway and suddenly your GPS warns you that traffic is ahead, you want to avoid it if you can. And so you can recalculate your route. And a router network, in addition to having information about where packets should go or where data should go, there's also sort of this general pulse on the state of its local network. What's going to happen if I send it down this path versus this path?

And so in light of heavy traffic situations on the network, maybe things will get routed a more inefficient way or a more generally inefficient way, because if we go the regular way, there's going to be a lot of traffic. The highway is completely jammed. So maybe what we'll do is instead take side roads, which ordinarily would take a lot more time, but no one's really using those side roads. And so we can route our packets that way.

So not every packet of a big chunk of data might take the same path from the beginning to the end. And our network becomes a lot more responsive if our router tables allow for there to be multiple options for where to go. We're not depending on that one truck moving out of the way. We can get off the highway at the next exit and take a different path. And so the Internet Protocol sort of does a little bit of that, too.

So that's the basics of the Internet Protocol. But there's one more issue to deal with, which is what happens if we do drop a packet? How do we know we're going to send that packet again? Right? Well, Internet Protocol doesn't guarantee delivery. We're going to be depending on another protocol to deal with that called Transmission Control Protocol, TCP. And we're going to talk about Transmission Control Protocol in the next video. I'm Doug Lloyd. This is CS50.

**TCP**

DOUG LLOYD: If you've been watching these videos in the order which we recommend, we're about to undergo bit of a culture shift. Because now, we're going to start talking about the internet and web technologies. So up until now, we've really been doing a lot of C. And when we've been running our programs, we have been running them from the command line. That's pretty much how the users have been interacting with the programs that we write. They pick something to prompt, something happens in the terminal window, and then it's done.

Sometimes you might have persistent data that remains afterwards. But that's pretty much it. It's at the command line. It's the only way the user can interact. From this point forward, we're going to start transitioning so that the users can interact with our websites. So we're going to be writing websites, which aren't written in C, but are written in a variety of other programming languages, including PHP, and it's sort of helper languages, HTML, CSS, and the like. So we're going to start talking about those things.

Before we get into web programming itself, I think it's probably a good idea to take a step back and talk about how computers and humans interact over the web. So this video is really a primer, a basic guide, to the internet. Now, the caveat here is the CS50 is not a networking class. So what we're going to be talking about here is pretty high level. We're not going to get into any low level details of how all this stuff works. If you're interested in that, I'd strongly recommend taking a class on computer networking. And we might even tell white lie or two just for the purposes of making the general understanding clear.

So with that said, let's talk about how we interact with the internet. So here we are. Here's us. We're pretty looking forward to getting onto the internet, which as we all know, is chock full of cats.

Now do we just connect to the internet like this? Well, probably not. Intuitively, you know that, say for example, when you change your Wi-Fi network on your computer, you don't see one called internet unless that just so happens to be the name of your local Wi-Fi. Right?

It's usually something like home. Or if you're at work, it might be the name of your company. There's not just one option called internet. And so something or some things exist in between when we want to connect to the internet. What are some of those things? Well, we're going to talk about that. We're also going to talk about some of the important things we need in order to be able to connect to the internet. And the first of these things is an IP address. So you've probably heard the term IP address before. What does it mean? Well, an IP address is basically a unique identifier of your computer on a network. Just like every home or office has a unique address to which one could send a mail.

Similarly, every computer if it wants to receive data or send data, needs to have a unique address. So that when information is sent or received, it's being sent from or received to the correct location. This addressing scheme, as I said, is called IP addressing. IP is stands for Internet Protocol, which we'll talk about again shortly.

Now, what does IP addressing look like? Well, the scheme basically was, when it was first implemented, to give every computer a unique 32-bit address. That's a lot of bits. That's 4 billion addresses.

And generally, instead of using hexadecimal notation, which we've used previously in the context of pointers in C to talk about addresses, we usually represent IP addresses in a little bit more of a human friendly way, representing them as four clusters of 8 bits represented as decimal numbers. Because humans don't frequently speak hexadecimal, unless you're programming. But people who use the internet aren't necessarily programmers.

And so making it easy and accessible for them to be able to talk about what their IP address is in case they maybe need to call up somebody to troubleshoot something, it's better to make it in the more common conventional decimal number format. And so an IP address just looks pretty much like this, w.x.y.z, where each one of those letters represents a non-negative value in the range of 0 to 255. Recall that an 8-bit number can hold 256 distinct values.

And so that's why our range is 0 to 255. And we have four clusters of 8 bits for a grand total of 32 bits. And so an IP address might look something like this. This is sort of a generic default IP address, 123.45.67.89. All of them are in the range of 0 to 255, so that's a valid IP address.

Here at Harvard University, all of our IP addresses start with 140.247. That's just the way that the IP addresses in this geographic area have been assigned. And so this might be an IP address that might exist here at Harvard.

So as I said, if every IP address is 32 bits, we have about 4 billion to give out, a little more than 4 billion. But we can kind of see a problem, right? What's the world population right now?

Well, it's somewhere north of 7 billion people. And in the Western world at least, most people have more than one device capable of internet connectivity. I have one right here. And I have another one in my pocket. And I have one back in my office.

And so that's three. And that doesn't even count the ones that I have at home, too. And so that's kind of a problem, right? We have at least 7 billion people and only 4 billion addresses.

And every device is supposed to be uniquely identified. We have developed some workarounds to deal with this problem, something called a private IP address, which we're not going to get into in this video. But basically, it allows further the web, the internet, to kind of fake out a little bit that you have a unique address by having private addresses and then funneling them through one single address, which is shared by many different computers.

But that's really not a long term fix. Even that fixed isn't going to last forever. And so we need to have a different way of dealing with this.

So as I said, we had about 4 billion. But that's not going to be good enough, right? And so the way that it has been decided there we're going to deal with this is to make longer IP addresses. Instead of 32-bit addresses, we're going to have 128-bit addresses. So instead of 4 billion addresses, we're going to have that huge number of addresses, which is 340 billion billion billion billion, so a lot of IP addresses.

And this new scheme is called IPv6 is commonly how it's referred. The old scheme being IPv4. It's a bit of a problem in that this problem has been known about for a really long time.

And you'll see this a lot in the context of computers and computing. We're good at anticipating problems. But we're bad at dealing with them even though we know about them. So IPv6 has been around for a while. And only in the last couple years have we actually started phasing in these IPv6 addresses to phase out the IPv4 addresses. But some places do have them. And they look similar to a regular IP address. But they are a lot longer.

So instead of now having four clusters of 8 bytes for your address, we now have eight clusters of 16 bytes. And 8 times 16 is 128. And we represent these in the less conventional hexadecimal form. Because having 16-bit numbers means that instead of being a range of 0 to 255, We'd have a range of 0 to 65,535.

And so having a bunch of those stuck together would be very difficult to read. And so we usually use hex just out of convenience. And so a typical IPv6 address might look something like this.

It's certainly a lot longer than the IPv4 address we've seen before. But this would be a valid IPv6 address. This one is also about IPv6 address.

This one happens to belong to Google. And notice there's a bunch of zeros there. Sometimes these addresses can get so long. And since we're still pretty early in IPv6, sometimes there can be big chunks of zeros in there that we don't need.

If you're reading this out loud, it's 2001.4860.4860.0.0.0.0.8844. It's kind of a lot, right? So if you see a bunch of zeros, you might sometimes see an IPv6 address like this, where they omit the zeros and use a double colon instead. This is OK, though. Because we know that there are supposed to be eight distinct chunks. And so by implication, we see four. So we know that there must be four sets of zeros like this, that fill it in.

So sometimes, you might see an IPv6 address not having eight separated chunks like we do here. You might see it looking like this. And that just means that everything you don't see in between where that double colon is is just zero separated.

So, OK. We know a little bit more about IP addresses now. But how do we get them? We can't just pick the one we want. If we did that, we might end up fighting somebody for the same IP address. Or somebody might have chosen it previously. If we try and take it, we're going to run into a bit of a problem. And so we can't just pick the IP address that we want.

So the way that we get an IP address is somewhere between our computer and the internet, that big internet out there, there's something called a DHCP server, a Dynamic Host Configuration Protocol server. It's a big mouthful of text. But really all it does is it assigns you an IP address.

Your DHCP server has a list of addresses that it can validly assign. And it gives you one. That's pretty much all there is to it. Now before DHCP, this task of assigning addresses fell to a system administrator. So an actual person would have to manually assign your computer and address when you connected to a network. So DHCP just sort of automates this process of giving you an IP address. But that's how you get it. It's just a program running somewhere between you and the internet that has a bank of IP addresses that it can give out. And when you connect to the network, it gives you one. So let's revisit this diagram. Somewhere between you and the internet, there's a DHCP server. OK. So that's good. Now, let's talk about DNS. So we've talked although these IP addresses. And we know that if we're going to uniquely identify a device on the internet, it has to have a unique address.

And we could visit that address if we wanted to. But you've probably never typed in something like 192.168.1.0 into your browser, right? You don't type in numbers into your browser. You usually type in human readable names like google.com or cs50.harvard.edu, right?

Those aren't IP addresses, though. So exists this service called the Domain Name System, DNS, that translates IP addresses to human comprehensible words or phrases that are much more memorable than remembering a set of four numbers or, soon, a set of eight hexadecimal numbers. That would be really challenging, right?

Think about before the days of cell phones. You had your memorize your friend's phone numbers. It might have gotten tough after a little while. And similarly, if you want to visit a bunch of websites, you probably don't want to remember a bunch of numbers. You'd rather remember a bunch of words.

So this mapping, this translating, of sets of numbers to human readable names kind of makes DNS the yellow pages of the web. And you can think about it as if it's just a huge list running from 0.0.0.0 all the way down to 255.255.255.255, which would be the highest possible-- that's the full range from 0s to 255s of all 4 billion-ish IPv4 addresses. I made up the ones on the top and the bottom. But the one in the middle there is actually an IP address. So if we visited 74.125.202.138, apparently that translates to that site there, io-- what the heck is that? Well, not every name that maps is actually clear what it is, right?

So sometimes somebody who owns an IP address might name their host something that they're actually not. For example, that IP address if you went there, is actually just google.com. But Google has a lot of different servers.

And they can't call them all google.com. So they have their own internal system for translating google.com to whatever server actually is connected to that IP address. And then there's another system that exists between to translate that gobbledygook here to google.com. But we won't get into that.

And similarly for IPv6s, we're also going to have a yellow pages that'll be a lot bigger. And similarly, in the middle there-- it was tough to find an IPv6 address that was legitimate. But I found one for Google.

But it's Google's Irish website. But if you went to that IPv6 address, if your browser was IPv6 capable, that would bring you to Google's Irish homepage. So there you go.

But this isn't entirely true, right? This the system seems cumbersome, right? If there's a huge list of 4 billion things to have to look up, that's pretty big. There's no yellow pages of the world, right? If you still get the yellow pages delivered to you-- I got mine the other day, and I just recycled it. But if you do get the yellow pages delivered to you, you don't get a book that's every phone number that exists on the planet, right? You get a list of the local phone numbers, the ones you're most likely to call.

And that's actually what DNS is. If you think about it, DNS is really the local yellow pages. And large DNS servers like google.coms, they are actually just more like libraries that have a copy of all of the local yellow pages or all of the local DNS records. So there's really no one repository of the full DNS of the internet, just like there's no one yellow pages of the world.

There are all these local small scale DNSs that exist out there. And there are services that aggregate them together. But they depend on those smaller DNS systems updating their information, so that they have the most accurate information.

So again, this analogy is large aggregating DNS systems are like libraries that have a copy of every yellow pages of the world. They don't themselves update those books. They depend on the books coming in, so they can update the information if they need it.

So the DNS system is not a giant block. It's decentralized across many, many servers. So now we know that somewhere between us and the internet there exists a DNS server as well as a DHCP server.

Now, access points, what our access points? Well, access points you're probably pretty familiar with from actually connecting to the internet. That's the network that you choose, the home or your work network or what have you.

And I'm generalizing the concept of an access point here for purposes of this video. But there are actually a lot of things that can be rolled up into access points. There are concepts of routers, which is sort of a general term that we use.

But there are also switches and things actually called access points that are separate from this general concept of an access point. But basically what happens is with IPv4, I said we have this concept of private addresses, right? And instead of every machine having a unique IP address, which we have run out of, because we're over 4 billion devices trying to connect to the internet, what we do is instead assign an IP address to a router. That router or access point just in your home, for example.

And the router's job as to sort of act as a traffic cop, allowing everybody who's connected to that router to use the same IP address to get out. Does that make sense? So everybody at your home has a private IP address. They can't connect to the internet, or the internet rather can't speak to them, through that private address. They can only speak to them through the address in the router. And it's the router's job to take information that you're sending the router and direct it to the correct place and for information that's coming into the router for the router to send it to you.

So the routers are really the devices here-- particularly a router in your home, the most common sort of usage case for most people-- that has the public IP address. That's the device that's connected to the internet. And you connect to the router to have information flow through it on your behalf.

As I said, a modern home network, the router and switch and access point are all kind of bundled up into a single device. Sometimes a modem is bundled in there as well. That's usually just called a router. But it's really all of those things together.

Large scale business networks or so-called Wide Area Networks, WANS, actually keep these devices separate. They have a switch. They have routers. They have multiple access points.

For example, at a university you'll see things that look like so-called routers mounted are all around campus. Those are all access points that flow into routers, switches, et cetera, to pass information along. Because these networks are so big that one single access point can't cover its large area.

And so these large networks, business networks, et cetera, split these into separate devices, so the network and scale and grow if needed. So again, somewhere between us and the internet, we have an access point. And that's what we connect to. And through there, we can get to the internet. As I said at the beginning of this video, this is not a course on networking. So this is not the entire story. And I've kind of glossed over it. And maybe I've left you even a little bit confused as to what some of these things are. But that's OK.

We don't need the whole story. It's enough for us to know moving forward just basically a little bit about how the internet works. So what we know is we have these private networks at our house.

And we connect to a router. And that router is connected to the internet at large. But what is the internet at large? I keep saying this, but what is it?

Well, it's really just all these individual networks at my house, and at your house, and at every other house, that are connected together. It's an interconnected network, an inter-net. So instead of thinking about the internet as this giant cloud, this ethereal thing that exists out there, it's really just a connection among all of these networks.

So here we go. We have our local network. And we're not the only person probably on our local network trying to use the internet. There's probably several of us trying to get in.

And we're not the only network that exists in the world, right? There are other networks, too, that are trying to connect to the internet. But the internet is not, again, a separate entity.

It's just a set of rules that allow these networks, these small networks, the blue, the purple, and the red network here, to communicate with each other. So there's no thing they're all connecting to. They're all just connected to each other, right?

And so somewhere on these networks exists the services that we actually want. So maybe in the blue network is where Google lives. And in the purple network is where Facebook lives. And in the red network, well, maybe that's where all those cats are.

And so if we want to get information about cats, we just traverse this chain of networks to get the information we want. And here, I've represented the network as all being able to talk to each other. And we can only talk to the network. But the network can't talk back to us.

But that's not true either, right? This is all a two-way street. Information can flow through networks back and forth.

How does it do that? Well, the internet's really a system of protocols. And we're going to start talking about what those protocols are in future videos.

But again, the internet is not a separate thing. It's a set of rules that defines how networks communicate, these small networks, these local network that we're used to, the people in our house, the people at our school, the people at our job, all sharing a network. And how these networks interconnect and talk to each other, that's actually what the internet's all about. So let's, in a future video, talk about some of the protocols that comprise the internet to hopefully give you a bit more of a well-rounded understanding. I'm Doug Lloyd. This is CS50.

**Hypertext Transfer Protocol, or HTTP**

SPEAKER: Let's talk about another protocol-- the Hypertext Transfer Protocol, or HTTP. So we've talked about IP and TCP in previous videos. And those are protocols that dictate how information moves from machine to machine and from program to program or service to service via the internet, via this network of routers and machines.

But that's usually not the entire picture, right? Usually when we send information, the program itself-- when data is received, say, for example, in email via TCP port 25 or a web page request via port 80, there's usually a system of rules there to process what I've just received. And HTTP is an example of just such a protocol.

HTTP is the only application layer protocol that we're going to talk about. But it is another set of rules dictating how information is to be transmitted and processed via the internet. In particular, HTTP specifies exactly how one must make a request for a web page and exactly how a server, a machine that hosts web pages, delivers that information back to clients.

So this protocol doesn't actually have anything to do with how information moves from point A to point B. It's really the system of rules for-- it's basically the rules of engagement for working with a web page, similar to when somebody waves their hand at you, you're supposed to wave back. That's sort of a conventional human protocol. HTTP protocol just says, if you want to request a web page, make sure your format looks like this-- sort of like formatting a business letter, for example. And the response will similarly come according to this protocol. There are other application layer protocols that we're not going to talk about in videos. But these include things like the File Transfer Protocol, Simple Mail Transfer Protocol for sending emails, the Data Distribution Service, Remote Desktop Protocol, RDP, which is used if you want to remotely access your computer from another computer, XMPP, which is frequently known as Jabber or chat, so this is the protocol for using chat services. And there are many, many, many others.

So every time you're using a service, the service is expecting information to be received-- a request to be received-- in a very particular format and is required to return information back in a very particular format as well.

So let's go back to our illustration of us wanting to talk to the internet. So we're happy, and we want to go to cats.com, right? So if we're just talking to cats.com, we might say something like hey, can I see your home page? And cats.com will probably respond, yeah, sure. Here you go. So that's a human sort of ask-and-answer.

What does that look like in HTTP? Well, it actually kind of translates pretty cleanly to something like this. We might say GET/HTTP/1.1 from host cats.com. So basically what I'm doing here is asking for the web page www.cats.com/. We usually omit the slash nowadays, but that would just mean cats.com's homepage.

Oh, and by the way, I'm going to be using HTTP version 1.1 to communicate with you. That's sort of analogous to saying, like, by the way, I'm going to be speaking in French, or by the way, I'm going to be speaking in English. That's just the format of the protocol. It's also 1.0, which is not commonly used anymore. So I'm speaking HTTP 1.1, and I would like www.cats.com/. Please get that for me.

And then there's other information, too-- the dot, dot, dot there, which is information about who you are so cats.com would know where to send it. But these are the two sort of critical parts at the very beginning of an HTTP request-- just like when you start a letter you say, dear, blank. This is very similar in spirit to that.

And if cats.com is going to say, oh, sure, here you go. They might respond like this-- I'm also responding. I also speak HTTP 1.1. Your request is approved, 200 OK. What you're about to receive is HTML and then dot, dot, dot some extra information. And at the very bottom of the request is actually the HTML, the markup language, the content of cats.com's homepage.

So HTTP/1.1-- I acknowledge your request was accepted via HTTP 1.1. Your request was approved. I can give you what you want, 200 OK. You're about to receive HTML. And then here's the HTML that you requested.

But sometimes our requests don't always go quite according to plan. Can I see your cats.html page? Well, what if they say, we don't have a cats.html page, which seems kind of unrealistic because they're cats.com. You'd think they would have cats.html. But OK. So this is sort of the conventional human interaction we've now had with cats.com. How does that translate?

This might be something familiar to you. Our request looked exactly the same, except instead of getting slash we're now getting cats.html. So now what basically this entire request is saying is please give me www.cats.com/cats.html. So the host and the middle part of that top line there indicate precisely what page I am asking for. But cats.com in this case isn't going to be able to respond positively. They don't know we're talking about. And so this is something you might have seen before-- HTTP 1.1 404 Not Found. I couldn't find what you were asking for. By the way, I'm going to give you back some HTML, and usually that HTML is the content of some 404 page. And in the case of cats.com, it's probably some cute cats in a basket with a sad 404 face next to them, because you're going to be sad when you don't get page that you were looking for.

That's kind of the basics of what a protocol, the HTTP protocol requests look like. They're really similar to how we would make a similar interaction in just human conventions asking for something and getting it back or writing a letter and expecting a response letter in a particular format. That's pretty much what HTTP is just canonicalizing for all devices that wish to access web pages, hypertext transfers.

So a line of the form, this the method request target HTTP version, is called an HTTP request line. It's usually the first thing that is transmitted as part of an HTTP request or if you're asking for HTTP. It's sort of like, as I said, saying dear, blank at the top your letter. They know that you're writing them a letter. So this is very similar to saying, I know that they're making an HTTP request and this is the particular format they're asking for.

HTTP version is probably always going to be HTTP/1/1. 1.0 also exists but isn't really used anymore. For purposes of CS50, GET is probably always what you're going to be using when you're actually making direct HTTP requests. But POST is another option that we're not going to talk about right now. And then request-target is what page on the host's server you would like to get. As I said, that host name is a separate line, usually the second line of the overall request. And so taken together, the host name and the request target specify a specific resource being sought. In our 404 example a second ago, I was asking again for www.cats.com, cats.com being the host. And in my request line, I said /cats.html. That was my request target. So overall I was asking for the contents or the resource located at www.cats.com/cats.html.

And then based on whether the resource exists and whether the server can deliver the resource pursuant to the client's request, you might get various status codes back. Some of these status codes you've seen because they're part of the response. Some of them, 200 OK, are probably pretty silent. You've probably never seen a page respond 200 OK. You just get the page. It's not like a 404 error, which is usually pretty clear. You usually see that it says 404.

So let's talk about what some of those status codes might be. Again, when the server responds to us, they're going to respond HTTP version status. Usually HTTP/1.1. What are these status codes going to be? Well, we might get a success. So in the success category, we might get code 200 with the text OK. What does this mean? Well, everything is good. You made a valid request. Here's a valid response. I was able to deliver exactly what you wanted.

Sometimes you might get other things that you won't notice right away but are somewhat failures. They're called redirections. There's two common ones here. 301 Moved Permanently-- what this basically means is the page is now at a new location. It will live there forever. And most browsers will automatically redirect you. So you'll never really see a 301, either, unless you're using a really out-of-date browser, possibly, because the 301 response is part of the dot, dot, dot of the 301 response. It also tells you where the new page is. And so most browsers will just redirect you there, assuming that you want to go there.

Sometimes you'll also get 302 Found. And this one you actually might still see occasionally. Sometimes pages move temporarily. So it's not going to be built into the request telling the browser to permanently change any time it sees the request that you make to change it to something else. So you might see 302 Found, which basically says this page lives somewhere else. But it's not going to live there forever. It will eventually probably go back to where you think it is.

Then you'll get things like client errors. So these are ones you've probably seen, now. You probably haven't seen the 200s or the 300s, but you're probably familiar with the 400s. And that's what we'll talk about in a second, 500s as well.

You might see 401 Unauthorized. Usually this means you're trying to access a page, but you haven't logged in. So you try and go to some profile or something on Facebook or you try and access some-- you're at work. You're trying to access something on your work's internet, but you're not logged in. You can't see the page. You might get a 401 unauthorized, which means we probably will be able to satisfy this request, but first you need to log in to do so.

Conversely, you might get 403 Forbidden, which is it doesn't really matter if you're logged in or not. This request isn't allowed. The resource exists on the server. But you are not allowed to access it. This is usually internal files that live on the server for various reasons but are not intended to be accessed from the outside world, and so they are forbidden. They live there. I'm not saying I can't find it. But I'm saying I cannot give it to you. And it doesn't matter if you're logged in or not. And then of course, the very common 404 Not Found. The file doesn't exist on the server. I would like to satisfy your request, but I can't.

You also sometimes see server errors, the most common generally being 500 Internal Server Error, which doesn't actually tell you anything at all about what has gone wrong. But it's not actually you making a mistake in your request. It's actually the server failing to deliver on the request somehow. So 500 is the general response.

You'll also see something like Service Unavailable, which I believe is code 503. And Gateway Timeout-- if you ever had a page just sit there loading and loading and loading and you never know if it's going to load and then eventually it just says-- just gives up. That's a 504 Gateway Timeout. The server wanted to execute your request, but something went wrong on the server side-- not on your side-- to cause that to be a problem. Now, we could end the story here, but what I'm actually going to do now is I'm going to open up my browser and show you how you might be able to see some of these status codes even if you don't generally see them. And we're going to do that by taking a look at some developer tools.

All right So here I am now in my browser window. And I want to learn a little bit more about these HTTP requests. How do I know-- certainly we know if a page goes-- when something goes wrong, we get a 404. We've all seen that. We don't need to illustrate that. But what are some other ones? And how would we see these requests in action?

So first thing I'm going to do is open up Developer Tools. So Developer Tools are built into most modern browsers and allow us to see things that we don't otherwise see-- some extra information sort of being transmitted underneath our web requests. I'm using Google Chrome here. And to open Developer Tools in Chrome, you just hit F-12, and it's going to open it up on the side. Once I type the request, I'll zoom in so we can see what's going on here. But what I'm going to do in my browser bar is-- and I'll zoom in over here-- I will make a request to www.google.com. We've all probably made this request before. I'm going to hit Enter.

Now, over here in my Developer Tools, I've chosen the Network tab. And you notice a lot of things here. Look at these-- 200 OK, 200 OK, some of these status codes coming up. I don't know why I'm getting 302 Found. I didn't realize I'd see that one. But basically notice that pretty much, in terms of my Google request-- I made a very simple request for Google's page. And in the process of delivering my request, Google has apparently made a lot of other requests on my behalf.

But I've made a get request for Google's page and I'm getting a lot of 200 OKs. I'm not seeing 200 OK on my screen, but I'm getting a lot of requests that have been made. One more that I'm pretty sure is going to work is-- for those of you who are really old-school, you may know that Facebook was not always at Facebook.com. In its early days it was at wwww.thefacebook.com. They apparently could not get access to Facebook.com for quite awhile.

And so what I'm expecting here is to get information. And we'll see if this pans out. What I'm expecting here is to get information that Facebook has moved permanently from thefacebook.com to Facebook.com. So I'm expecting somewhere near the top of my requests over in my Developer Tools to get a 301 notification that Facebook has moved permanently. Again, I won't see 301 on my browser screen. And because it's a 301, it's a permanent move. My browser, being that it's a modern browser, is probably going to redirect me to Facebook.com anyway. But let's see what happens.

And now I'm going to go to thefacebook.com. And yep, there it is right at the top. It went away, but it was there. Let me scroll up here. Right here at the top. I made a request to thefacebook.com, and I'm getting a response that this page has moved permanently. And then 307 here is an internal redirect. And so this is what has actually moved me to the much more familiar www.facebook.com.

So these response codes do still happen, even if we don't see them. I'm not going to illustrate 401, 403, 404, because you've probably seen those at various points. And 500, I would just be kind of-- we'd get lucky if got a 500 because we don't know what servers are currently down anywhere. But these codes do exist, and there is a way to access them even if we don't see them firsthand on our systems. I'm Doug Lloyd. This is CS50.

**Hyper Text Markup Language (HTML)**

DOUG LLOYD: So we spent about-- if my math is right, and I think looking back-- I think we spent about 35 videos talking about various aspects of C, maybe a little more, maybe a little less. And we didn't cover everything in C, but we covered a big chunk of the language, the vast majority of it, certainly for common uses. Now we're going to talk about another language, HTML. And we're going to cover it in just one video.

But that's going to be OK. That's going to actually become something you're going to get used to. Now that you have the fundamentals of one language, it's actually pretty easy to start learning others. So we're going to start to step a little back and gloss over the basic differences between these languages and sort of leave you to it. There's a lot of really great resources on the internet, which we're going to start directing you towards because the internet is a vast repository of information. And so it's not like you'll be losing out necessarily by not having the information covered in a video. You'll still be able to get everything you need and use the knowledge you've already built up by understanding C to make the learning curve for these other languages actually a lot flatter. I promise.

But let's talk about one language that's really fundamental for every web page, which is HTML. HTML is the Hyper Text Markup Language. HTML is a language but it is not a programming language.

HTML doesn't have variables. It doesn't have logic or functions or anything like that. We can't do any programming per se in HTML. Sometimes you'll hear people describe themselves as HTML programmers, which isn't entirely accurate. We can't write HTML programs.

HTML is just used to mark up text. It's called a markup language. And what this does-- this markup-- we use tags in HTML and these tags-- this markup-- semantically defines the structure of a page and causes the plain text that exists between tags to be interpreted by browsers in different ways. And perhaps it's best to explain this by way of an illustration.

Here's a very simple HTML page, not an HTML program, again, an HTML page. And we know it's an HTML page because we've bounded everything with HTML tags. So this is what an HTML tag looks like. It's between angle brackets. And notice at the top we have HTML and at the very bottom, after we've done what is apparently a lot of other HTML, we have angle bracket slash HTML. So that sort of is the boundary between what is HTML and what is not. And of course, conventionally, just as you wrote all of your C programs with dot C extensions, all of your HTML files will end with dot HTML extensions. But there's more going on here. We don't just have these HTML tags. We apparently have this thing called a head tag. Well, OK, what's that?

Well maybe it's best to distinguish by way of a body, body being the content of the web page. So maybe the head tag defines stuff that isn't in the browser window proper, but is somehow important to our web page being rendered correctly. For example, inside of the head tag we have title tags.

So title being hello world, that's actually going to be what shows up in the tab in Chrome or in safari or Firefox-- whatever browser you prefer-- that's what's going to show up in the title. And before tabs it would show up in your entire browser window and you can only have one page open in a browser window at a time. So that's going to be the title of my page up in the tab or the browser window bar, hello world. And then the content of my web page will be world, hello. So let's take a look at what some thing like this might look like. This is a pretty simple HTML page. So I'm here in my CS50 IDE and I've zoomed in a little bit. And I'm just going to open up hello dot HTML and show you that this is pretty much the page content that we saw before. My simple HTML, head tags, title tags, body, and so on. I've indented to be clean.

And then what I can do in my IDE is just preview the page. And there we go. The content of my page is world, hello, and I don't see anything in from the head tags there. It's just the content of the body. World, hello. And again the body just said, world, hello. The other part is missing.

So that's really all it is. This is a very simple basic HTML page. Now I've indented my HTML to be really nice and organized, but I don't actually have to. I could make it look pretty ugly. And this would still work. This would be the exact same web page. I've just gotten rid of all of the white space.

As it turns out, white space is data. And so when we're sending data from sender to receiver, from server to client, data costs money. And so getting rid of whitespace is actually a good idea if you're somebody who serves up a lot of web content. It's a bad idea if you're somebody who's learning this stuff and you want to have it nicely organized. This is a lot easier to parse than this. But it's functionally identical.

The indentation and stuff like that doesn't actually matter in HTML. All that matters is opening tags and closing tags in the correct order. Notice what happened here, though. The markup gives us a way to communicate extra information about what we've written. The Hello, World part was interpreted as the title. And the world, hello part was interpreted as the content or what should be visible on my web page.

There are over 100 of these different tags and lots of great resources online to find them. We're going to talk about a few of them in this video, some of the really fundamental stuff. But we're not going talk about it all because it would be exhaustive to do so.

Another thing you can do, though, is open up developer tools. And if you recall from our video on HTTP, I explained how to open up developer tools. In Chrome it's usually the F12 key to open up the developer toolbar. Then instead of choosing the Network tab, you can choose the Elements tab. And if you load a web page, you'll actually see the HTML that creates that web page. And so you can learn a lot about HTML by looking at your favorite websites and seeing how they build the various pieces of them that you like. So maybe there's this cool pattern or something like that. How do they make it with HTML? Well you can just open up your developer tools and hover over that element and see exactly what HTML makes it. So that's a really good way to learn HTML, and I strongly recommend that you do it both to learn HTML and also to learn a little bit about some of the options available to you in developer tools, which will certainly come in handy as you begin doing more intensive web programming.

So let's take a look at a couple of common HTML tags. And we'll jump and take a look at what these tags will also render as by looking at some files in my IDE. So here are three very basic tags for tweaking the visual appearance of text. There's B tags, I tags, and U tags. And respectively what they do is render the text between them in bold, italics, and underlining. So let's see what that would look like on an actual web page in my IDE.

So here in my IDE I have a file called BIU dot HTML. BIU dot HTML just being bold, italics, underlining. I'll open it up.

And we'll see that here I have this text is B tags bold. This text is I tags italic. And this text is U tags underlined. What is this going to look like? Well again, all I have to do is go over here to my browser, my file browser, click Preview, and this is what comes up.

The text in between the B tags is indeed now bold. The text in between the I tags is indeed now italic. And the text in between the U tags is indeed now underlined. So that's pretty good. We now know how to make text look a little more fancy or draw emphasis to certain things. Another couple of common tags here are paragraph tags, P, and header tags, which I've rendered here as HX.

These P tags, these paragraph tags, break your text up into paragraphs. It's not enough to just hit Enter and leave spaces, because a computer is only going to do what you tell it to do and it ignores white space for the most part. So we can't just hit Enter and expect our computer to interpret that we want to start a new paragraph. We have to very explicitly say this is one paragraph-- this is another-- by enclosing each in a set of P tags.

And we also have these options for H tags, these header tags. We have six different levels of headers, one, two, three, four, five, and six, which are progressively larger and larger headers. And they get smaller and smaller and smaller and smaller. So we have a top level header, a second level header, and so on, and so on.

Let's take a look at maybe some P tags and some header tags in action on a web page. So here in my IDE I have a file called PH dot HTML, PH being paragraphs and header tags. Open that up. There's a lot going on here because I've put some lorem ipsum, some just random text in here. So I'll zoom out a little bit because there's so much going on. But notice that I have at the very top here I have an H1, a level one, header tag. Then I have a paragraph, which is just a bunch of random text-- lorem ipsum-- just default standard filling in text. So I have two paragraphs inside of that level one header and then down below I have a level two header here on line 24, a second level header, and another two paragraphs. Well what does this look like if I view it in my preview? Let's see.

So notice that the first level header here is actually quite a bit bigger than the second level header. So we used H1 tags. And notice that the P tags allow us to break things out into paragraphs. If we had gotten rid of those P tags and actually just put Enters or Returns in between what we hoped would be the different paragraphs, they would all just slam together and it wouldn't have this nice paragraph separation with space above and below. And so that's what paragraph tags and header tags are commonly used to do to draw attention to portions of our web page in that way.

Next up are some tags that we use to build lists on our web page. So we have unordered lists-- ULs-- which are just bulleted lists, ordered list which are numbered-- OLs-- and inside of either one of those we need to have sets of how to indicate list items, LI. And so we have open UL tag and we put items inside of it. And then when we're done with that, we can close the UL tag.

And similarly we can have an ordered or numbered list and put list items inside of that. So let's take a look at a couple of lists and what they would render as on CS50 IDE. So I have here in my IDE a file called lists dot HTML. Let's take a look.

And notice here I have an unordered list with five things in it. And then I have an ordered list, and I've changed the tag a little bit, right? I've said start equals six. It turns out with an ordered list I can set the starting point wherever I want-- by default it will be one-- by just adding this so-called attribute to my OL tag. And so this list will start counting at six. So the elements of that numbered list should be six, seven, eight, nine, ten, because there are five elements in the list, as opposed to one, two, three, four, five, which would be the case if I had said OL without specifying the start attribute.

So we'll just preview this so you can get a sense for what's going on here. And there we go. There's my list. The first five elements are unordered or bulleted lists. And the next five elements are a separate ordered list starting from six. So that's how we can build lists using HTML. Another thing you might want to do with HTML is build a table of information of rows and columns to present information in a particularly organized way. To do this with HTML we can have a table definition beginning open bracket table. And then inside of that table we might have a set of rows, TR tags to indicate each row. And then TD tags go inside of TR tags to specify a column within a row.

Why is it called TD and not TC? Well, TD stands for table data. Usually you're putting your information there. So that's why it's TD and not TC. It's a little bit confusing.

So you have table tags and inside of your table tags you have a number of rows, TRs. And inside each row you have TDs for the number of columns that you want to have in that particular row. Let's take a look at a very simple table over in CS50 IDE.

So I have here a file called table dot HTML. Let's have a look at what that looks like. There's a lot going on here but if you notice I have a table open. I'm starting the definition with table. And then in my first row I apparently have four columns, one, two, three, four. And then I'm done with that row.

Then I start another row and do two, four, six, eight. Finish that row. Do another row, three, six, nine, 12. And then a last row, four, eight, 12, and though it's a little cut off here, 16.

I finished that row. I finished the table. And then I'm done with my HTML. What does this look like? Well, it's not really much to see. I've clearly organized my information in a somewhat more organized way. But it's not super pretty here. And we're going to deal with that when we talk about CSS. We'll revisit this idea of what we do to make a table-- maybe format it a little bit better? But I do still have four rows, each of which has four columns, and really what this amounts to is a very simple four by four multiplication table.

Just a few more tags we'll talk about. Let's talk about the concept of an HTML form. So you may have seen this in the context of logging into a web page. Usually you type in your user name. You type in your password, and you're good to go. That would be the beginning of a form.

Skipping over div a second. We also have inputs which kind of fit inside of forms. These are the elements that you're actually typing into, or the radio buttons you're ticking, or the check boxes that you're ticking off. So these go inside of forms. And they comprise basically each row of the form if your form is formatted well. Then there's this concept of a div, which doesn't really fit in any particular category of tags like the ones I've been doing previously. It just sort of demarcates the beginning of some arbitrary division-- div-- of the page. There's no visual break. There's no line. It's not set off as a separate chunk automatically. You'd have to style it that way to do that.

It just sort of says I want a piece of space on my web page, and I'm just going to call it this division of my page. We can put stuff inside of divs, and in fact, when we head over to IDE in a second, we'll see that I'm putting my form inside of a div.

So I have here in my IDE a file called div form dot HTML. Let's open it up. Notice that like I said, div is kind of arbitrary. Right? It doesn't really mean anything. So I have an arbitrary first division of my page. And then instead of another div later on, starting on line eight, I have this form. And inside of the form I have a number of inputs, fields of the form. So I have a field whose name is A-- which doesn't really mean anything right now-- that apparently takes text, another one that takes a password, another that's a Radio button, another that's a check box, and another that's a Submit button. Well, what does this all actually look like? Well, let's take a look. We'll open it up in our preview window. Notice that this arbitrary first division-- there's no visual separation here. It didn't really do anything, right?

And then I have my form. And I didn't do any special formatting. So the form is just one big row of information. If I had formatted my form differently, I might have it line by line by line. But I didn't do any styling. Again, we're not talking about CSS here. We're just talking about HTML.

Well in my text form I can type-- remember that forms of type text so I can put my name. And in my password I can type my password. And because that field is of type password, you don't know what my password is. It's all dots.

I can also choose to tick off a radio button or tick off a check box. Or I could submit my form. And I didn't do anything, so when I submit my form, the page just refreshes. But I could perhaps configure my Submit button to do something else. And we'll see what we can do with that in a future video on PHP. But this builds a very simple form that we can use to have users interact and submit information to our website.

One last comment before we move on to some other tags are to take a look at this input tag one more time. Notice that I've highlighted the ends of the tag in red. Every other tag we've seen so far has had a beginning and an end, an opening tag and a closing tag.

But an input tag doesn't. There's no text that goes in between input tags. All of the information we're intending to convey is bound up as part of the attributes of that input. Notice we have input name equals x. Type equals y. That's really all the information we need.

This is called a self closing tag. It doesn't require an opening and a close because all of the information is contained inside the tag and its attributes. So sometimes you'll see this, too. So just be aware that if you have a tag that is entirely self-contained, it opens and closes itself with the open angle bracket on the left and the slash angle bracket on the right. We'll see another one of those right now with image tags as well.

Before we talk about images, we need to talk about hyperlinks. If we want our web page to be interactive and move us around, it would be nice be able to click on one of those what has typically been a blue link. This is actually how we build a hyperlink in our web page. And interestingly enough there's another HTML tag called link, which is not a hyperlink. A here stands for anchor, and that's how we indicate a hyperlink.

A href equals x means go to web page X. And everything between the open A tag and the close A tag is what's going to be that underlined blue text that looks like a link that we're familiar with. Below that we have an image tag, which is a self closing tag for displaying an image located at X. And you might be able to change that image by specifying width and height and other attributes in that dot dot dot there.

At the very bottom here we have a very interesting looking tag that doesn't have a closing tag. It's exclamation point doctype HTML. So HTML has been around since the early 1990s for building web pages, and it's gone undergone several revisions since then. Most recently in 2014 it underwent a revision called HTML5 which is now the current sort of de facto HTML standard.

To indicate that our web pages are written using HTML5, this is how we start off. It can be omitted but what that basically means is you can't use any of the tags that are HTML5 tags, those new tags. So we always start off if we're using HTML5. And all the tags we've talked about previously are not HTML5 tags. But this would indicate that HTML5 tags will be present. And so we have exclamation doctype HTML, which is at the very beginning of our HTML file, and then after that point we actually have our HTML open tag and proceed from there.

The last one is a comment tag, which looks slightly different, too. It starts off with angle bracket exclamation dash dash but no closing bracket. In between those two elements there is where you write your comments. And let's take a look at images and comments and links in CS50 IDE.

So I have here a file called image link dot HTML which I'm going to open up. And notice I've got a couple of comments here in my HTML comments. So just like in C and other programming languages, HTML just by being a markup language does have the ability to have comments. And so I'm apparently going to place a picture of Rick Astley somewhere between this div tag, this arbitrary division. Apparently that file is located at Rick dot JPEG, which if we head back over to my file tree for a second, is a file that exists in the current directory. So that's OK. I can reference it.

Then I can have internal links. So notice on line 11 here my href is hello dot HTML. So that just refers to hello dot HTML which exists in the current directory. And I can also have external links by just specifying HTTPS to indicate that I'm not talking about a file in my current directory. I'm talking about a file that exists somewhere on the internet, which I have to request using the HTTP protocol.

So let's take a look at what this page might look like and get ready for a picture of Rick Astley to show up on your screen. So I'll preview this. There's Rick Astley at the very top in this arbitrary division I put it at the top. And then down below I have my links, right?

I have a link to hello dot HTML. And if I click that, I get moved over to this page that we're very familiar with from the very beginning of our program. If I pop that page open again, if I pop image link open one more time, I can also go externally to CS50's website. And there we see-- I'll zoom out a little bit here-- we'll see CS50's website sort of embedded in the middle of our page. So I was able to make an internal link as well as an external link.

The last rule with HTML that we're going to talk about here is that your HTML should be well formed. In C we talked a lot about the various syntax of things. In HTML the syntax really revolves around tags. Every tag you open needs to be closed. And in fact, every tag you open should be closed in reverse order.

So if you open a bold tag, an italic tag, and then an underline tag to do all three to a particular set of text, you should close them in reverse order. So if you opened bold, italic, underline, you want to close underline, italic, bold. This sort of encapsulation is what keeps HTML nice and organized.

Unlike C, though, syntax errors won't actually cripple your HTML possibly. Your HTML may be not well formed but would still work. And so these errors can sort of slide by. It's up to you to really be vigilant. Sometimes they will fail but sometimes you can get away with it.

It can be a really difficult task, though, to keep track of when you opened a tag, when you closed it, especially as your HTML files get bigger and bigger. You'll want some help. And there are online validator tools that you can use to have a look at your web page and see if it's well formed HTML. And you should definitely take a look at those and start to use them as you begin doing some work with HTML, writing HTML, just so you get some good habits about organizing your HTML in a good way and good style and making sure that you're not doing anything that could create a syntax error that would cause you a bit of a problem down the road.

I'm Doug Lloyd. This is CS50.

**CSS**

DOUG LLOYD: So let's take one more video to talk about one more language. This time we'll talk about CSS. So CSS, which is short for Cascading Style Sheets, is another language we use when constructing websites. Think about it like this. If HTML is what we use to organize the content we want to put on our page, CSS is the tool that we generally use to customize how our websites look, and how the user experience really is, interacting with our website.

Similar to HTML, CSS is not a programming language. It doesn't have logic. It doesn't have variables. It doesn't have any sort of that flow related things that C does. It's a styling language. And its syntax is pretty simple, and just describes how the elements of our page have certain HTML elements should be modified. To that end, if you haven't yet watched our video on HTML, or are unfamiliar with HTML generally, you may want to take a look at that first, because this discussion of CSS is going to depend on some knowledge of HTML.

So here's a really simple CSS stylesheet. Even if you've never programmed with CSS before, I'm pretty sure you can figure out exactly what this stylesheet does. What does it do? Well, applied to the body of our web page, everything between body tags on our HTML, and it sets the background color of that page to blue. Well, it's a very simple stylesheet. It's actually very human friendly language, CSS. So even if you've never used it before, you probably could guess what that did. In fact, if we loaded a page, where this stylesheet was embedded somehow, the background color of our page would be blue, and not the standard white.

So how do we build stylesheets? Well first, we have to identify a selector. In the last example, that selector was body. Then we have an open curly brace, and we're going to begin defining the stylesheet for that selector. In between the curly braces, we just have a list of key value pairs. The previous value pair was background color blue semicolon, but we could do more and more of these. You could have multiple things applying to that tag, that selector body. Each one of them is separated by a semicolon, and we call each one of them a declaration, a CSS declaration. When we're done with all the styling we want to apply to that particular tag, we just have a closing curly brace to end the stylesheet, and we're done defining the stylesheet for that particular selector.

What are some common CSS properties? Well, maybe you want to put a border around something. So you can say, border, that would be your key, and then your values would be, what style, color, and width you want it to be. So the style could be a solid line, a dotted line, a dashed line, a ridge line, which would be wavy line. Maybe you want to have it be blue or black or green. Maybe you want it to be 1 or 2 or 10 pixels wide. You can specify all of those things. Maybe you want to set the background color of your page in a particular way. We already saw that, setting the background of the body to be blue.

Then you can use a key word, so CSS has certain colors that are built into it, blue, green, black, a very simple colors we know. But you can also specify any hex color that you'd like. Recall that colors can be identified by a set of three hex numbers from 0 to 255, rg and b, the red, green, and blue component. And so we can specify any color we want by, instead of using blue or green or black, using pound and then six digits of hex, and that would give us the six digit color. So that's the background color.

We also have the foreground color, which is usually going to be the text of your page. And you could similarly do that with key word and or six digit hex. So you can specify any color you want for the text of your page against a particular background color, up above. You can also change and deal with font, and the way text is rendered on the page.

So you can change your font size. You can use key words, such as extra, extra small, or xx small, or medium, large, and so on. You can use fixed points, 10 point, 12 point, and so on. You can use percentages, 80%, 20%, where 100% is the default font size, which is usually going to be something like 11 or 12 points. Or you can even base it off of the most recent font size. If you just wrote something and you know what you want is for it to be smaller, but you don't know exactly what size you want it to be, well, you can just say, font size smaller. And it will base off of the, just up above, it's font size. And you can get smaller or larger. So there's a lot of different ways to specify font size.

You can also specify what font family you want. If you have a particular name, there's a way in CSS-- we're not going to talk about it here-- to define a very specific font and embed it into your page. You can also use generic names. There's a lot of web safe fonts that are pre-defined in CSS. And if you are a user of Microsoft Office in the last 20 years, you're probably familiar with a lot of these web safe fonts already, Times New Roman, Arial, Courier New, Georgia, Tahoma, Verdana, and so on. Those are all considered web safe fonts. And actually, part of the reason they came to be was to be used to make web-- every page had access to this default set of fonts with various serifs, and all this font stuff we won't get into, but these are usually accessible in your CSS, even if you don't otherwise define the fonts. Lastly, you can align your text, instead of it being, by default, aligned to the left, you could align it to the right, or you could align it centered, or justified so that it hit both margins. So those are all options you can use to change what your text looks like, and how it's displayed on your page.

Your selectors don't have to be tags only. We previously saw a body tag selector, and tag selector does look just like that. You name a tag, and then you define a stylesheet for that tag. But you can also do something called an ID selector. An ID selector looks pretty similar. But notice, that now I'm not using an HTML tag, I'm using, in this case, #unique, or hash unique. If you recall from our video on HTML, we talked about how tags can have attributes.

And one attribute that applies to pretty much all HTML tags, but we didn't talk about it, is something called an ID tag. So this particular CSS would apply only to HTML tag that has a very specific ID, that you've named. So if you have somewhere in your code, somewhere in your HTML file, a tag and you specify as an attribute to that tag, ID equals unique, this particular stylesheet here will only apply in between that tag with the ID of unique.

You can also do something called a class selector. So in addition to having ID attributes, you can also add a class attribute to HTML tags. And when you use a class attribute, it can be applied to multiple tags. So if you have several things that are similar, maybe you want to say, open tag blah, blah, blah, blah, class equals students. And then this particular stylesheet would apply to every tag whose class is students. In this case, we'd set the background color to yellow, and we'd set opacity, which is a tag we didn't talk about, but just deals with how transparent something is, to 70%, in this case.

There's two options for writing stylesheets. You can write them directly into your HTML by placing the stylesheets in between style tags. And those style tags go inside of the head tags of your web page. The perhaps more preferred way to do it is to write a separate .css file, and then link it into your document using link tags. Link tags, again, are different from hyperlinks, if you recall from our video an HTML. And link tags are how we pull in separate files. It sort of like the equivalent of the #include for web programming.

So let's take a look at table.HTML. If you recall from our HTML video, I showed an example of a very simple multiplication table that looked pretty ugly, and maybe there's a way to make it better with CSS, to make it actually look more like a multiplication table, or something that isn't just stuck together with no actual division between the rows and the columns. So let's head over to CS50 IDE, and take a look at how CSS can, sort of, tweak what we started with before, and make it something a lot better.

So we're in CS50 IDE now, and if unfamiliar, we'll pull up in this table that HTML page. Table.HTML basically just defines the contents of a multiple-- it was supposed to be a four by four multiplication table. It's pretty straightforward. And we would think that it would look pretty well organized. But in fact, when we preview this page, we see that it's kind of ugly, right? Clearly we have rows and columns here. There some sort of separation. But it's not a meaningful separation. We're not actually getting too much information here. And there's no separation between the rows and columns in terms of horizontal or vertical rules. We could probably make this look a little bit better. So let's try. So I'm going to close this tab up here. And I'm going to close my table.HTML, and I have another version here called table2.HTML. We'll open that up. The body of the page is pretty much the same, but I've changed a little bit at the top. And I'll scroll up here. Notice that this time, I'm using embedded style tags. I've opened a style tag, and I'm now defining a CSS stylesheet just inside of it. I have a stylesheet that says, table. That's my tag selector. I'm not using dot or hash, which I would be doing if I was using an ID or a class selector. I have a tag selector here-- table. This style is going to apply to every table tag. Apparently I want to put a one pixel wide, solid blue border, inside my table. That sounds like it would probably help the situation, right? We're going to have things look a lot better. So this is fine. Stylistically, I've just embedded my stylesheet in here. It's certainly an acceptable way to do it. Let's see what this looks like. So I'll go back down here, and I'll will preview my table2.HTML. Well, that's not quite what I wanted, but it is exactly what we asked for. We said that this style is going to apply to our table. Our table now has a one pixel wide, solid blue border around it. We're not actually getting at the table cells. We're just getting at the table. So CSS worked. It has applied a stylesheet to our table. But didn't quite do what we wanted it to do. How do we get to do what we want it to do?

Well, let's take a look at one more version of this in table3.HTML. So I'm just going to close these tabs. I'm going to go back over here to my file tree, and open up table3.HTML. Again, it's going to look pretty similar here at the beginning. But notice, this time, instead of using a stylesheet embedded right in there, I'm going to link in a stylesheet using the link tag. So I'm apparently linking in a stylesheet called tables.CSS, and this well equals stylesheet just means-- well, what is this file relative to what I'm doing-- is a stylesheet that I'm using for my page. So if I really want to see what I'm doing with the CSS here, I need to go open that table.CSS file as well. So we'll go back over here again to our file tree. There's table.CSS. We'll pop it open. Now we're seeing a little of the CSS. Apparently, I have a couple of things going on here. I apparently want to put a five pixel wide, solid red border, around my table. We already know that that's going to just go on the perimeter. We saw that in table2.HTML. With each row, I apparently want to specify that the row height is 50 pixels high. So for every row, remember that's what the tr tag does, I'm making it 50 pixels high.

Lastly, I have this comment. And this is how we make comments in CSS. It's very similar to seize block comments syntax slash star. All the text you want. There's no slash slash though in CSS. For short inline comments, we have to use this particular format here. It looks like I'm doing a lot of things in my td tags. Remember td tags, or table data, which really are just the columns inside of our rows, and apparently for each piece of data in my table, I want to set the background color to be black, the color to be white, color is foreground color. So this is going to be the text of my page. I want big font, 22 point font, and I want it to be of the font family, Georgia. So I'm not going to have the default font. I'm going to specify Georgia, which is one of those web safe fonts that we've seen before. I want my text to be aligned centrally, in the middle of the box, I don't want it to be left aligned or right aligned. And I want my column width to be 50 pixels wide as well. Let's take a look at what this looks like, and see if this is maybe an improvement. So I'm going to go to table3.HTML, which recall, includes table.CSS as a link, and we'll preview it. That's a lot better, right? This is actually starting to look a lot more like a multiplication table. I have that red border around my table but now I also have specified that each row is 50 pixels wide, or it's 50 pixels tall-- excuse me-- each column is 50 pixels wide. The data in each column, and only the data, has a black background. So that's why those white lines are there. Because the space in between all of those cells, it's not a border in and of itself, it's just I'm only filling in the cells, which actually makes the borders of the table, which apparently did exist all along, it was just thin white lines. Now they're visible. Now they pop off on the screen. And so this is a very simple stylesheet that I've applied, and now my table looks a lot more like a four by four multiplication table, instead of a just jumbled mess, where everything is clearly rows and columns, but not super well organized. We're really just scratching the surface of what you can do with CSS here. Fortunately, the CSS documentation is pretty straightforward. You'll use several of its attributes, fairly frequently. The ones we talked about earlier in this video. There are several that you probably won't use all. And that's fine, too. But just know, that there's a lot of documentation out there. So even if we didn't cover everything, you're certainly not left on your own. But CSS is really fun to experiment with. And I would strongly encourage you to play around with your web pages and see how you can make them look and feel to improve the user experience of visiting your page. I'm Doug Lloyd. This is CS50.

**JavaScript (JS)**

[INSTRUMENTAL MUSIC PLAYING] DOUG LLOYD: So near CS50's end, we introduce you to several different programming languages, and we aren't able to cover them in nearly as much depth as we're able to cover C earlier in the course. The goal here is not to give you a complete rundown of what these languages do, but to give you enough of the basics and show you enough of the parallels to things that you already know so that you can go forth and really flesh out your knowledge of these different languages. And another one of these we're going to cover right now is JavaScript. Now, similar to other languages we've discussed, it is a modern programming language that is derived from C's syntax. So if you're familiar with C, you're going to be able to pick up JavaScript pretty quickly. It's been around just about as long as Python. It was invented a few years afterwards in the mid 1990s. Now what's really important about JavaScript, as well as HTML and CSS, which we've discussed in other videos, is that those three languages pretty much make up the core user experience of what it means to be online. So understanding how to use and work with JavaScript, HTML, and CSS will allow you to really create websites that users will enjoy visiting, because you'll be able to create a great user experience for them with these three languages. To write JavaScript, it's really easy. Open up a file with a dot js file extension, and start writing in JavaScript right away. If you're familiar with a language like PHP, you know that you have to sort of bind your code up in these delimiters-- bracket, question mark, PHP, and all that stuff. We don't have to do that in JavaScript. We literally just start writing our code in a JavaScript file, which is pretty nice. Our website will also know that what our file is is JavaScript, because we're going to tell it in an HTML tag. We're going to say, I have a script that is of type JavaScript, and it is a file with a dot js file extension. Pretty straightforward. Now, unlike Python, which runs on the server side-- that is, on the side that is hosting the website, JavaScript applications run client side on your own machine when you are visiting a website. To include JavaScript in your HTML, it's pretty easy. Just like when we include CSS with style tags, we can include JavaScript in between script tags. So we can have an open script tag and literally start writing JavaScript right in our HTML, which is kind of cool. But also like CSS, where we have link tags, we can write our JavaScript in a different file and connect it using the source attribute of a script tag. In CS50, and really probably more generally, this is definitely the preferred way to go, particularly if you have a JavaScript file that might be used-- or the contents of which might be used-- on different pages of your website. You don't have to write the JavaScript between the script tags on every single page. You want to be able to just refer to an outside file and link that in every single time. So let's quickly run down some of the fundamentals that you might use in JavaScript. So, variables-- they're really similar to Python variables and really similar to C variables. No type specifier is required. And if you want a local variable, you preface it with the var keyword. So in Python, we would say something like this-- x equals 44. We don't want to do that in JavaScript, because that would actually create a global variable, if you were to put a semicolon at the end. We would want this-- var x equals 44. That creates a local variable called x in JavaScript, and stores the value 44 in it. Conditionals-- all of the old favorites from C are available for you to use. If, else if, else, switch, question mark colon-- all of those are fair game. We're not going into detail on those at all, because we've covered them in our videos on C. But they're all here to use in JavaScript, and behave pretty much exactly the same. Loops, meanwhile-- all of ones we're familiar with from C are also still available-- while loops, do-while loops, and for loops, although we do have a couple of other variations that we'll talk about in just a little bit. Now, functions in JavaScript are all introduced with the function keyword. So it's a function. Write a function name and parameters, and then we define our function between curly braces just like we would in C, except in C, of course, we're not using the function keyword. But there's also a catch with JavaScript, which is that functions can be anonymous. Meaning you don't have to give them a name. Now, you might be asking yourself, wait, how can I call a function If it doesn't have a name? Well, we'll talk about that in just a little bit, and we'll talk about what I mean in particular here when I'm describing binding things to HTML elements. We'll talk about that in the video on the document object model, but we'll talk about anonymous functions in just another minute. I'll give you an example and you can see how they might be useful in certain contexts. And those contexts actually do kind of materialize a lot in JavaScript programming. JavaScript, like C, and like Python, are analogous to Python's lists. Has arrays, and you can declare an array. It's pretty straightforward. For example, var nums creates a local array. And then I have square brackets, and I just have a comma-separated list of all the values that I want to put in the array. I can also mix types. I don't have to have all integers, or all floats, or all strings, like I do in C. I can have different types mixed together there. JavaScript has the ability to behave a few different ways. That's why it can be a little bit tricky as the very first language to learn, because a sets up a bunch of rules for itself and then breaks them. It is very, very flexible, but perhaps almost a little too flexible as a first language. But it can behave as an object-oriented programming language. Object-oriented might be a term that you've heard before, but if not, we'll try to give a quick little crash course here. An object is really similar in spirit to a structure that we are familiar with, hopefully, from C. Now, in C, structures can have a number of different fields or members. Another term for those that we see commonly in object-oriented programming are properties. But those properties, or fields, or members, just like in C, can never stand on their own. So for example, if we define the structure for a car like this-- struct car, and inside of that, there are two different fields or members-- int year and char model tens, or a 10-character string that we can use to store the cars' model. We can do things like this. We can say struct car Herbie, which declares a variable of type struct car named Herbie, and then we can say Herbie dot year equals 1963. Hurbie dot model equals Beetle. That's totally fine. That's valid C. But we can never say this in C-- we can never say, year equals 1963, model equals Beetle. Now, we can't say that because year and model are so fundamental to what it means to be a struct car. They are part of a struct car. And so if we're ever using year and model-- we can't use them in the abstract. We always have to associate those things with a particular struct car. So that is not OK. Objects, in addition to having fields, or members, or properties, also have something called methods, which is sort of like if a C structure had a function that could only ever apply to that structure. So it's as if we have a function where structures are the only things that need to be passed in. Now, that, we have in C. But we can't define a function inside of the curly braces of a struct. That is more-- that is object-oriented programming. That is different from here. But just like properties, methods don't stand on their own either. It's not like a function that just exists in the abstract. It is a function that only exists in the context of, in this case, an object. So where we might do this in C-- function parentheses object, where object is a parameter to the function being called-- that is not object-oriented styling. Object oriented means the object is at the core of everything, and instead, we're going to call this function that is a part of the object. OK? So this is what object-oriented programming looks like in a very, very basic form, is, there are functions that are affiliated or associated with objects, and we call those functions as follows-- by specifying the object and saying we want some function to execute on that object, like this. The fields and methods of an object are really similar in spirit to the idea of a dictionary, which you may be familiar with or recall from Python. So we could, for example, have something like this-- var Herbie equals-- curly braces, now, not square brackets, it's curly braces-- and then, a comma-separated list of key value pairs, where year 1963 is sort of like saying Herbie dot year equals 1963 in C, and Herbie dot model equals Beetle. This is akin to how we would do the same thing in JavaScript, by creating an object and giving it a couple of properties right away. All right. Now, let's go back to loops, because I mentioned earlier, there are a couple of different new loops that we can use in JavaScript. So now that we have this object, how would we maybe iterate across all of the key value pairs of that object? Or in fact, how we iterate across all of the elements of an array, if we wanted to do that, besides using a for loop, a while loop, or a do-while loop? So in Python, we saw something like this-- for key in list, and then we would have some code where we would use key every point from there on down-- a stand in for the i-th element of the list. You can't do that in JavaScript, but we can do something pretty similar for var key in object. And then open curly brace. And between those curly braces, we can use object square bracket key to refer to the key-th element of the object, or the key-th key in a key value pair set of the object. We also have a slightly different variation for var key of object, which, instead of having to use object square bracket key, we can now refer to iterating across all of the values-- as opposed to this, which iterates across all the keys, this would iterate across the values. So we can cover both sides of a key value pair using for var key in, or for var key of. Here's an example of an array where we're going to use these two different types of loop. So we have here a week array that has 7 elements in it-- Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday. And I want to use a four var day in week array. So I want to use a for in loop, and I want to console dog log. Console dot log is the JavaScript equivalent of print f. You can access it using developer tools in most modern browsers. What's going to happen when they print this out? It's going to print out a list of all of the keys. Well, in this array, there's really no keys-- there's a bunch of indices. So this will print out 0, 1, 2, 3, 4, 5, 6, because there are seven elements of my array, and I am logging which element I'm talking about. If I wanted to print out the days instead, I would use a for of loop, and I would get the following printed out to the console-- Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday. What if I want to concatenate information? So remember, we can tie different strings together. We could do that in Python. We can do it as well here in JavaScript. And we do that by using plus. So if I want to console dot log a bunch of different strings concatenated together, I just use plus to do it. But here's a little bit of a catch, right? If I'm using plus, now it means two different things. I want to concatenate strings together, but maybe here, I want to add. I really encourage you to take a look at what happens if you write the week array into some JavaScript, and then have a loop that iterates across and console dot logs this. You might find that you get something that you don't expect. And so, JavaScript is making its best guess, and it's assuming that day and 1 are both strings, when really, what I want to do is add them together, literally, because they're numbers. But fortunately JavaScript gives us a way around that by using a function called parse int. Now, this is going to give us what we expect to see. This, on the other hand, is going to give us something we don't expect to see. But it's just, again, a case of, we're using the same operator to mean two different things, and the language sometimes has to make a best guess. So it's not always a great thing that languages nowadays don't often require us to specify a data type-- if we were able to specify a data type here, we would be able to disambiguate this. But here with JavaScript, it thinks it's being very helpful for us by not specifying a type. And it's going to make our best guess as to what we want to do, but sometimes it guesses wrong. But it gives us ways to work around it if it does indeed guess wrong. So arrays are actually a special case of an object. And, actually, in JavaScript, everything is a special case of an object. X, any variable, var x equals 44-- that's an object that just happens to have one property. Arrays are an object that happens to have a number of properties, and also some certain methods that can be applied to them. So we can call the size method, or pop, or push, or shift. And I'll leave you to figure out, or to do some research into, what those different things are. But suffice it to say that they are functions that are basically so baked into what it means to be an array that we can just call them without having to write them in the first place. That's pretty cool. Here's another cool thing that you can do with arrays-- there's another method for them called map, which basically allows us to apply a function to every single element of an array. And in particular, this is a great example for using an anonymous function. So let's take a look at how we might use an anonymous function and why it might be useful to do so. Here's an array called nums. It has five elements in it-- 1, 2, 3, 4, and 5. Now, what I want to do is, I want to map some function on to them. So what I'm saying here is, nums equals-- so I'm going to do something to nums. I'm going to reassign that back to nums. Nums equals nums dot map-- and remember, map takes in a function. It's a function that I want to apply to all of the elements of that array. One thing I could do is just write this function somewhere else, and give it a name, say I called it-- what's going to happen here is, every value's going to be doubled. Let's say that I wrote a function called double numbers. I could say, nums equals nums dot map parentheses double numbers, and it would just double every number. That would be fine. But here, notice I'm using that function key word, and I'm giving a list of parameters-- in this case, num-- but I'm not giving that function a name. Why am I not giving that function a name? Well, I'm just doing this mapping here. I'm never going to need to double these numbers again. Why am I going to clutter up the name space, and give it some name when I'm never going to use it again outside of this context? It's kind of convenient that we can actually use the function in this case, then, without having to give it a name, and still have it do what we want to do. So all I'm doing here inside of those parentheses of maps is, the open parentheses on the first line there-- and that close parentheses is actually on the third line. And in between the parentheses, where I'm specifying the parameter to map, I'm literally defining an entire function in there. And then if I execute this line of code, what's going to happen to nums? Well, you can probably guess, it's going to just double every element of nums. All right, one more thing I want to touch on really quick about JavaScript, and that is the notion of events. So an event is something that goes hand in hand with HTML on JavaScript, and it is a response to a user doing something on a web page. So for example, perhaps they click a button. Perhaps the page has finished loading after the user hits refresh, for example. Maybe they hover over a portion of the page, or something like that. All of those things are events. And JavaScript has support for what is called an event handler, which is some JavaScript code that will execute once that event has taken place. So it allows our site page to be very interactive. We type in a box, for example-- we could have an event handler, that pops up a message that says, please enter your password. So it's just a complete-- it's an empty field that says nothing at all. You start to type in it, and then this box pops up because you started typing, which is an event on the page. And it tells you to do something that is a JavaScript event handler taking effect. Many elements of HTML have support for events as an attribute. So here's an example of some really basic HTML. I have my head and my body, and inside of the body, there are two buttons. And they have this attribute called on click. On click is a definition, basically, for an event handler. So I want something to happen when I click the buttons. Right now it's nothing, but I can put something in there so that when I click on either of those buttons, that function that's in between the quotation marks there would get called. We can write a really generic event handler in JavaScript, for example, that creates an event object, and will tell us which of those two buttons were clicked. And it might look something like this. Button on click equals alert name event. If we do this, we have basically set up our HTML, so that when either button one is clicked or button two is clicked, the alert name function will be called, and an event will be passed in. The event is automatically generated by the page, and it basically contains all of the information about what just happened. And from that information, we can extract more information. So for example, here what the alert name function might look like, and notice that here, I'm also accepting an event parameter. What am I doing then? I'm getting a new JavaScript local variable called trigger, and I'm asking for that event's source element. And what that is basically telling me is, what is the element on the page that was interacted with that triggered this event? Or rather, put another way, what element effectively was passed in to this function. Because, again, this is automatically generated. The page knows which button we touched, and so, basically, what's happening here is, that button-- whichever one of the two it was-- is getting passed into this function. Then-- and you'll see a little bit more of this when we talk about the document object model-- I can access that trigger's inner HTML to figure out which button was clicked. Now, what is inner HTML? Let me jump back a couple of slides for a second here. The inner HTML is what is between the button tags. In this case, button one is inner HTML and button two is also inner HTML. So what happens when I click on those buttons is, it will alert to me, you clicked on button one if I happened to click on button one, or, you clicked on button two, if you happened to click on button two. Now, again, we are just scratching the surface of JavaScript, but I wanted to give you a sense of some of the different things that you can do, so that you can then go forward, look at the documentation, and see the very wide range of things that you can do as well, by just applying some of the basic tenets that we've learned about in CS50, and really expanding your knowledge exponentially. I'm Doug Lloyd. This is CS50.