Introduction to the GitHub, Testing, and the DOM

Video Transcript

Video 1 – GitHub and Git Set Up

Okay, so let's sign up to GitHub and install Git and install our keys. So, what we need to do is first get a login to GitHub. So, we need to go to this site; then, we need to install Git. And to do that, we need to go to this site. And then these are the instructions for generating your SSH keys on your machine and registering them with the Git Agent, and that's this site. And then we need to go back to GitHub, uninstall our "public key" there. Okay, so let's first go to GitHub. So, here we are on GitHub, and we need to sign up. So, we're at GitHub.com. Sign up. So, create your account. So, you need to go through this, and then you'll have your account.

I've already got mine, So, let me sign in now; once you've got your account, you'll have your own little cloud if you like. I've got, if I look at my repos, I got quite a number of repos already. Okay, so now you've got your GitHub set up. We need to go back and get Git. So, we go here, and this is git-scm.com. And down here, you can install, and it's got Windows as well. So, to do that, we'll click on download for Mac. It's quite a straight forward install. It'll download. Here we go. So, I'll keep it. It's open it. I've already installed that (2) shows I've already got two copies.

And it's a straightforward install, just double-click. So, for Windows, it's going to be slightly different. But once you've done that, you should have Git installed. So, let me now go to, I'll open a terminal window. So, here we have our terminal window, and we can check that we've got 'git' doing '--version'. And you see, I've got 'git' '2.23'. Okay, so we've got 'git'. That's there. We've got '.ssh'. Now, we need to get our keys. Let's go back and see how to get the keys. Okay, we need to generate new SSH keys and add them to the ssh-agent.

So, this will take you through that. I'm going to copy this so we can generate keys. Let me go here. And after the '-C', I need to put my email address. Okay. Now I need to specify where I wanted to go. And this will be, put it under your username. So, mine is John Williams as it happens. And under '.ssh', and I need to give it a name. I'm going to call this 'test1'. Okay, so that'll be the name of my key. Don't enter any passphrase. Just enter return and return again, and we'll now have those keys.

So, if I do 'ls test1', you'll see that I've got 'test1', 'test1.' and there's one '.pub'. So, this will be my private key, and this will be my public key, and they're under '.ssh'. So, we've got them. Now we need to tell the agent that is running the Git Agent. We need to tell the agent about those keys. Now, let's go back to the instructions. So, here we've got our key pair, now add it to the agent. So, we need to make sure the agent is running. Yup, 'Agent', and that's its process ID. So, the agent is running fine.

Now, if we're on macOS, we need to configure our config file under '.ssh'. So, you'll need to edit it, and you need to change this. If it doesn't have anything there yet, you need to copy all of this. But then we need to put our key name. Let me go. I've already got this file, so let me, so here's my config file. I've got that. And you see, I've got a different name. What we need to do is edit and put test1 in there. I'm not going to change mine because I've already, using another key.

But let's now tell our agent about our key. And to do that, we do an ssh-add, 'ssh-add', and the key is 'test1'. So, that will add it to our agent. Okay, so now, Git knows about our private key. Now we've got to tell GitHub about our public key. And in particular, we need to take a copy of our public key. So, I'm going to 'cat test1.pub'. So, here it is. I am going to take a copy of all of that. So, I've got a copy.

Now, I need to go back to GitHub. So, here's my Repositories, GitHub. I need to go under Settings. And down here SSH and GPG keys. And I need to add a new key, call it test1, and cut and paste it in. There it is, add the key. Now, it should all work. What this will allow us to do is to, now it won't ask us for any security. Every message from Git will automatically be signed with my private key, and GitHub will recognize it.

Video 2 – Introduction to the GitHub Cycle

So, in this section, we're going to talk about the GitHub and Git cycle. So, GitHub is the Cloud version. So, you have repositories, repos in the Cloud, but you also have copies of them on your machine, and on your machine, we use Git to manipulate and keep track of things and typically will need to add file. So, we'll do a Git.add, for example, for new files that need, need to be tracked. And when we're comfortable that they're going to be part of our repo, we will do a Git commit. And finally, to communicate with the cloud, we do a Git push. Now, these functions have been provided to us now in VS Code.

And so, in VS Code, we'll show you how you do these Git adds, Git commits, and Git pushes that you'd normally do in your terminal window, but now you can do it in VS Code. And one of the things we'll look at is manipulating a div on a webpage. And we're going to inject posts into that div. So, we're going to take a number of; we're going to have an array of posts. And we want you to pull them out and inject them into a div on a webpage. So, this is about building the webpages and also controlling your repos so that you keep whatever you have on your machine in sync with what is in the Cloud, in GitHub in the cloud.

Video 3 – GitHub Cycle

So, let's take a quick look at how we're going to use GitHub. So, GitHub comes in two parts. There's the part that exists in the cloud which we're going to call GitHub. And there's the application on your laptop, which is called Git. And the way we're going to use it is that the staff

are going to have their own repo. And you might need to get a copy of, for example, a problem set or a homework. And so, to do that, you're going to go to the staff repo and click on fork. And what that's going to do is take a copy of that repo and reproduce it in your repo. So, you're going to have an exact copy of that in your repo. Now that's in the cloud. So, fork happens in the cloud, now the next stage is you need to get it down to your machine.

So, the way you get it down to your machine is that on your machine in your terminal window, you issue, you type in git clone and the name of your repo up here. So, it will be git clone and it'll be GitHub something, your name, your repo. Okay, and I'll show you how to do that. And now what happens is that we now have a copy of the repo on your machine and it's under git control. So, now there are three copies, or there is a copy in the staff repo. There's one in your repo in the cloud and you've got it on your machine. Okay, so let's suppose now on your machine, you start making some changes. I've added a little red line here to show that you've made changes. Now, you want to get your git repo in sync with the one in the cloud that's yours.

So, you do a git add. and that adds all the new files to, just locally. It tells git locally. Okay, I'm adding all of these files now into git and then you add a git commit and you give it a name because they're going to keep track of this in the cloud. So, you do a git commit -m, and then you give it some name. That makes sense. So, this will be my first changes. Now we need to get it back up into the cloud. So, now what we do is we do a git push and that'll take your copies now and put them into your repo. If you've got all your key set up as we showed you, that should be a very smooth and easy thing to do and it happens very quickly it'll synchronize these two. The normal way that you'll be working is that you'll be making some changes.

And let's suppose you make a mistake and you delete files. You can actually recover them by issuing a git pull. And that'll go back up to your repo and pull down the last version of whatever's up there. So, here I'm showing the changes now being the files that you deleted, repositioned on your own local machine. So, that's the usual way. And then obviously you'd make some other changes and then do a git add, a git commit, and a git push. So, that would then push them back up. So, there's the git pull, and now you've got them back in sync. Now again, you may want to ask the staff to take your changes, and you do what's called a pull request.

And you can't force your changes to the staff because they're they own this repo, their own repo. But you can say, "Look, I found some mistakes in what you gave me. I'm issuing a pull request to you." And if the staff agree, then they'll pull those changes that you made into their version. So, that's the full cycle. Let's take a look at it in action. Okay, so I've logged into my repo in GitHub, and here are my repos. Let's go and get one from the cloud here. Now, I know there's one called hello. So, let suppose this is the staff repo here. We can Fork it. So, now we're going to Fork it to my repo. So, you see it's calling it hello-1.

So, we've got it, and there's a number of files here. Now we're going to clone it down to our local machine. So, I take a copy of this to my clipboard. So, let's go down to my local machine. Here we are and now I need to get to the right directory. I'm under MIT, this is where I'm going to put it. And I issue a 'git clone' and the name of the repo. So, it's cloning it down. I'll go into that

directory. 'hello-1'. Here we are. Let's edit, for example, this 'index.html' file. Okay, so let's get my editor, and here's hello-1 and here's index.html. Let's make a change here and just put in here 'h1'. Let's call this 'Hello World'. Okay, and I'll save it. And now that it's saved, let's go to 'git' and do an 'add .', and I'm going to do a 'git status' just to see, and it's saying 'index.html' has been 'modified:'.

So, I want to push it now, backup, but I need to do a 'git commit -m' first. And I'll say 'Hello World Edit'. Okay, I've committed it. Now I do a 'git push' and it's pushing it back up to my repo in the cloud. So, let's go back up there. Okay, I'm back up in the cloud and I am in hello-1. I'm just going to refresh it and let's take a look. Here we go. Let's take a look at this index file. Now it should say Hello World in it. And here we are, Hello World. So, it's pushed the changes back up to the cloud. So, that's the cycle of cloning down to your machine, making changes, and doing a git add, a git commit, and a get push back up into the cloud. So, now we can make more changes down on our own machine. And we can repeat those last steps of git add, git commit, and a git push.

Video 4 – VS Code GitHub Integration

So, here I am in my repository, and this is my website. And let me clone it down just to go through the steps that we went through before. So, I'm taking a copy of this, and I can go down to my machine. Here I am on my local machine, and I do 'git clone', and the site. And it's cloning down. And I should now have the site, 'johntango.github.io'. So, there it is. And if I look in it, I've got, this is my website where I've got my picture, and 'index.html'. Let me just show you how we can manipulate this in Visual Studio.

So, here let's open that folder, it's under MIT, and here it is. And I've got two files in there. Let me just change this one. So, let me call this now 'John's Portfolio'. And I save it. And you'll notice here a little 1 comes up. So, we haven't talked about this tab, but this allows us to integrate from VS Code with GitHub. So, I'm going to click on that. And it shows me here that I've got one change waiting to be committed. So, let me commit it. Now, to commit it, I need to put a message here. So, let me call this 'port1', portfolio 1. And now I click on plus Now, notice down here at the bottom, there's this little circle.

Let me bring this up just a little so you can see it there. Now, when I click on plus, we will see one change staged. That now there's one file waiting to be uploaded. So, I clicked on plus; then I clicked this tick to commit. So, it's committed. And now it's waiting to be pushed. And I click on this little button, and it will automatically push up to the website. And now, the little 1 will be gone. And if we go back to our website, and I refresh this, that was p4. Now, let me refresh this. We'll see that now it's port1. And if I look inside here, John's Portfolio, Visual Studio Code integrates very well with GitHub.

Video 5 – Introduction to Testing

Testing is part of creating software. Testing is part of maintaining software. Testing is part of deploying software. Testing is part of securing software. I think you get the picture. Really, testing is part of the entire software development lifecycle. There's a great number of job opportunities simply within testing. We will start off with some basic examples here, we will show you some of the leading test suites, and we will leave it up to you to dig deeper into that space. At first, as you're getting things going, you don't really need to have the full suite and all of the components of testing. However, as you go forward and you start to work with teams, as you start to share your code with others. This is something that you will definitely need to have.

Video 6 – What is Testing?

In the past, software creation was widely accepted as the System Development Life Cycle model, where you started with requirements. Here you try to understand what you were going to build. You move to design your software architecture. Then you move to your code construction, construction, your implementation, then verification, your testing, and you ended up with maintenance and operation. Accompanying these views were models for testing, such as the one you see here, where you tested your small units of code at the beginning. Then you move to Design testing, Architecture Engineering testing, System Requirements testing, and ultimately, User Requirements testing.

And although today, software creation is very different, the question of testing is still central to software creation. In the market, you will see terms such as quality assurance and quality engineering, and more recently, intelligent testing, where you make use of the Cloud. You make use of big data. You make use of machine learning to accelerate your testing with automation and Cloud resources. In this course, we will start you with the basics. We will get you familiar with some testing suites using testing initially and eventually creating some simple tasks. If you want to learn more about the suite that we will be using, which is Jest, you can navigate to https://jestjs.io/.

Video 7 – Install Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is sole performant that what was originally part of a browser has become one of the most successful servers in the market; it powers a great deal of things beyond serving pages. It is also used for storage engines. It's used in the IoT community. And it has a very powerful, productive ecosystem that we will initially use here to be able to load some of our packages and do our testing. So, go ahead and navigate to the address you see before you https://node.js.org/ and download the version for your

operating system, pause the video and carry that out. Once you are done, open up a terminal window, and you can confirm the installation of node by entering 'node -v'. And if you get the version back, that means you have installed it successfully.

Video 8 – Testing Hello World Exercise

Testing software, given its complexity, has been part of creating software for a very long time, well over 40 years. And as you start to write code and your code grows, keeping a handle on what's taking place, keeping some certainty about what's happening within your code is something that you want to have. Because of it, testing suites and testing ecosystems have emerged around the practice of building software. And one of the approaches is to write your software even before you start to write your code; it gives you some sense of where you're going. It gives you some feedback even before you start writing your code.

Now, as you get started, this might be too much to have a handle on as you're trying to master a lot of the different technologies. However, as your code matured and time passes, this is something you want to come back to. In this approach that you're seeing before you, it's test-driven development. And in this approach, you write your tests even before you start to write your code. And the approach would be to write some test that would initially would fail. Then you will write some minimal code that would allow you to pass the test.

And then you would refactor, you would improve your code, and you will continue to do this until you had pretty good code that captured the intent that you were trying to represent in code. So, if we take a look at a number of steps, you would think about the following. You would write a test; you would watch it fail. You would write some more code; initially, it would fail for a few iterations, then you would fix it. It would eventually pass, and to the point that you felt good. And then you would add, commit, and push into your GitHub repository.

So, we've given you here a small "Hello, World!" testing repository. And what you're going to do is that you're going to fork, you're going to make a copy on to your own environment within GitHub. Then you're going to clone, make a copy onto your local environment. Then you're going to do your work. And once you're done, you're going to push your changes back up. So, in a more graphical representation of that workflow. We're going to go to GitHub. We're going to then make a copy on to our local environment, as you see here. Then we're going to work in a decentralized, remote way, that means we're going to work locally.

And then once we're done, and we have added our changes, we're going to go ahead and push those changes back into GitHub. So, we'll start off by navigating onto a repository that we have created for you, and the address is kogsio/greeting. Now, as you can see here, I am at an account, and this account happens to belong to Peter Parker. This is meant to be a learner's account, and this person, Peter Parker, is going to navigate to the repository that we just showed the address for. And he's going to go ahead and make a fork of that repository.

So, I'm going to go ahead and paste the address here, and I'm going to navigate. And you can see here that address for kogsio/greeting. And at this point, we're going to create a Fork, and we do that by pressing on the button here on the upper right-hand side. And you can see here in a moment that that Fork is being created. I am getting my own copy onto my own account; in this case, the account is Peter Parker. And you can see here that we have a small, a small indication of where that, where that repository came from.

It says, forked from kogsio/greeting. Next, we're going to go ahead and create a Clone, and we do that by pressing, clicking on the Clone button. And we're going to use, in this case, the SSH. So, we can set the permissions to be able to push back up. And you will only be able to do this for your own repository, your own account. And so, I'm going to go ahead and copy that. And as you can see now, I am at the Terminal, at the command line. And I'm going to go ahead and enter the command to clone that to my local machine. So, I enter 'git clone' plus the address that we got from GitHub, which is, you can see there the address it has 'git@github.com:peterparkerdata/greeting.git' So, I'm going to go ahead and clone.

And as you can see that they are, that is made, that has been copied down onto my local machine. And now, I'm going to go ahead and move into that repository, which is called 'greeting'. And if I 'clear' my screen and I list my files, you'll see there that I have the same files that I had in GitHub. And let me go ahead and show you that. And as you can see there, as I scroll down the page, we have the same files. We have hello; we hello.js, we have hello.task.js. Next, we're going to install all of the dependencies that are required for us to carry out our testing, and we do that using 'npm', which is part of node. And so, we will enter 'npm install'.

And so, at this point, we have everything that we need. I'm going to go ahead and 'clear' the screen. And I'm going to go ahead and run the tests. We run it by entering 'npm'. Once again, that is part of node, and in this case, we type 'test'. And I'm going to go ahead and run that test now. Now, as you can see there, that test was successful. I had one 'Test Suite', which is a grouping of test. And inside of that 'Test Suite', I had only one test, which is 'Tests'. And so, let's go ahead and take a look at the code.

As you can see here, I have the editor on my left-hand side. On the right-hand side, I have the Console, the Terminal, and I'm going to go ahead and walk through the code. On the very first line, I am bringing in the file ' "hello.js" '. Then I have a header, 'HTML page', and then a little bit of 'script' that is using or referencing a function that we have within hello.js. You'll see here that it calls a function call 'hello( );', and then it writes that response to the 'console'. And if we take a look at the hello.js file, you'll see that it's a simple function call 'hello( )' that returns ' Hello World!" '. And if we look at our 'test', you can see there that the tests in this part here says, 'expect(hello( )).toBe("Hello World!");'.

Now, this part, the hello.test.js file, is part of just a testing framework for JavaScript that we're using for our code. And you can see here also that there are some other utilities. We have something called 'course-utilities' that loads the code that you then reference here on 'load' and then gives you access to those objects or those functions within this environment. And so, you

can see there that there is some language to learn when it comes to testing, but for now, let's go ahead and simply modify the code so that it's expecting to have ' "Hello World!" '. And if I remove the exclamation point there, saved it.

And then I'm going to go ahead and rerun the tests. You'll see that we get something different. You can see there that something failed. And if we scroll back up, you'll see here that it said expected ' "Hello World!" ' with an exclamation point, and we received ' "Hello World" ' without it. And so, it gives you there a good amount of information. In this case, we had a single task, and that was, that meant that the entire test failed.

Normally, you had many more. So, I'm going to go ahead and add that exclamation point back in, save the file. And then I'm going to go ahead and rerun. And this time, as you can see, the tests pass. And it says here that one test that we were doing outputs the correct string is, in fact, correct. So, this is a minimal, a minimal test, a "Hello World!", where you are cloning, and you are checking to see that you have everything in place to be able to run the tests that you will use throughout the course.

Video 9 – MIT Fact Test Exercise

In this exercise, you're going to once again fork a repository, then clone it to your local environment. But in this case, you're going to have some tests that are not passing. It will be your job to fix the code. So, the tests pass. And once you're done to add your changes, and commit them, and push them back up on to your repo. Let's start at the Peter Parker account, who only has one repository, which was the Hello World that we did for greeting. Now, we're going to go ahead and navigate to the source repository, which is MIT Facts, and we're going to go ahead and clone it.

We're logged in, once again, as Peter Parker. You can see there that this is being forked now. And once that is done, we're going to go ahead and clone it to our local environment. Once again, just simply highlighting here that it is forked from kogsio/mitfacts. And let's go ahead and copy our address. We're now at the Terminal, and we're going to enter 'git', well, actually, let's go ahead and place this repository on our 'Desktop'. Now, we're going to enter 'git clone' and the address to our copy of that repository, the forked repository. So, you can see here that that is 'peterparker', 'mitfacts'. And I'm going to go ahead and hit return.

And as you can see, that has been cloned to my local environment. I'm now going to go ahead and move into my repository that I've just cloned or just 'mitfacts'. And inside of it, let me go ahead and 'clear' my screen. Inside of it, I'm going to run 'npm install' to get all of the dependencies. And as you can see there that has completed. Now, I can go ahead and run 'test'. And as you'll see, a number of tests fail. So, you can see here we have one 'Test Suites' that 'failed', that is the grouping of old tests. But we have '3' tests that 'failed', and '2' that 'passed' out of '5'. So, let's go ahead and scroll up.

And this is information about MIT. This is why it's called MIT facts. And you can see here at the very top that the suite called 'MIT'. And I have two tests that pass. The first one is 'Object properties' there, I'm just checking for properties, and I'll show you the code in a minute. Then I have 'City match' that is passing. But I have two, I mean, I have three that are not. I have 'Colors match' that failed, 'Founded range' that failed, and the 'Motto' that failed. All of these refer to MIT. So, let's go ahead and take a look at the code.

Once again, I have the editor and the Terminal side by side; you can see here that I have three files, mit.html, mit.test, and mit.test.js. The HTML one is simply confirming some functionality with the browser. You can load it into the browser and take a look at the console for each of those values. But the one that has the data, the one that we're testing for, is the mit.js. And you can see there that we have the 'city:', the 'colors:', the 'mascot:', the 'founded:' date, and the 'motto:' for MIT. And in the test file, we are testing for a few things.

The very first one, type 'checks' to see that that object has a certain number of properties, and we're testing for three properties, which is city, colors, mascot. And you can see there the syntax that is being used. And as mentioned before, if you want to know more about testing, you can navigate to Jest, the testing framework and it has all of this rich API well documented. The next test that we have is that we're testing for ' 'Cambridge' ', and then we're testing for ' 'Colors' ' of MIT. Then the ' 'Founded range' ', we're being generous, not a specific date but around the date.

And lastly, the ' 'Motto' '. What is the motto? And you can see there that it's looking for at least one part of the model, which is [inaudible]. So, let's go ahead and change one so that it passes. And we will change an easy one. We will change the founded date. In this case, we'll make it the actual founding date, which is '1861'. I'm going to go ahead and save that file. And now, I'm going to go ahead and rerun that test. And as you can see now, we only have two failures, whereas before, if I scroll up, it was three. And so, we pass one more test. And you can see here that out of the five, three are passing.

And so, I will leave the other two to you to fix and so that you can pass a 100% of the test and then push your code back up. But let's go ahead and finish that get working cycle before I finish here. Now, before we go ahead and push those changes back up, I wanted you to comment that checking in code that does not pass test. That's a pretty bad practice. But in this case, we're simply doing it to illustrate the completion of the workflow. So I'm going to go ahead and start off by checking the status. And as you can see there, the only file that has changed is 'mit.js', where we changed the date. And I'm going to go ahead and add that. And I will use the 'add' all command.

Now, if we check the 'status' again, you'll see that we have one change that has been staged, and now we're going to go ahead and commit it. Well, let me go ahead and 'clear' my screen first. Enter 'git commit -m', and I will say, 'fixed date'. We have committed that change, and now we're going to go ahead and 'push' those changes into GitHub. And as you can see there, the changes have been made. The warning that you see as that I am communicating to GitHub for the very first time. This is a clean machine, and so it's warning me that it has been added to the list of

hosts that I communicate with, meaning that it's not suspicious. But you can see there through the rest of the message that this change has been pushed into our GitHub repository.

Now, back on GitHub, you can see there Peter Parker accounts in the mitfacts repository. And if we scroll down to the listing of the files, we'll see here the comment that I made when I fixed the date. This turns out to be a very useful comment because as you look at the history, you will give yourself hints about the changes that you have made in the code. And so, if we click on this and we look at the code, we'll see here that 1861 is now that the date for that property on the MIT object. So, go ahead and try it yourself, fix the rest of the, fix the rest of the issues with the code, resolve the test failures, and go ahead and check that code back in.

Video 10 – The World Wide Web and Tim Berners-Lee

So, in this exercise, we're going to talk about the basic technologies that Tim Berners-Lee invented in the early 1990s. He was at Cern, and they were producing lots of data in the collider there, and researchers around the world needed to get access to that data. But it was difficult because to get access to a file on some remote machine was not at all easy, that you needed to know the exact name of the file. You have permissions that you needed to get onto the machine and get access to that file. So, he invented a number of technologies. One is called HTML. It's the markup language of webpages.

So, this is how we make it easy for you to understand how the data should look. And along with HTML were CSS style sheets that came along. But he also invented a protocol so that machines could talk to other machines and get data for us, and that was HTTP, Hypertext Transfer Protocol. An HTTP has simple commands like GET, and POST, and UPDATE, and DELETE that allow us to manipulate these documents on the web. Now, another big breakthrough was having links to documents. So, within a document, we can link to other documents. The idea of hyperlinks was already there before Tim Berners-Lee, but he embedded it into the World Wide Web.

So, some of the things that we'll be doing is following links and pulling down the data that they're linked to. So, he basically came up with an amazing method for machines to talk to other machines. And you need to understand some of those basic technologies. URLs, for example, where we have http://www. say, mit.edu is the name of a machine. It's a friendly name. But machines are really identified by IP addresses. And to get an IP address from a URL, we need a special server on the web, and that's called a DNS server, Domain Name Services server.

And what that does is allow URLs to be translated into IP addresses. Now, because on a single machine, there may be multiple applications running, we actually need to talk to a specific application, and they're identified by what we call ports. So, often you will see after the http, you know, mit.edu, you might see colon and a number, like 80. Now, 80 is the default port for web servers, but they can also exist on other ports. And so, you need to understand these basic mechanisms, for how to address machines, how to get data from them.

And this was a big revolution that eventually gave rise to the Internet of things, where machines talk regularly to other machines and move data around the web. So, this is an important lesson about the basic technologies of the web. And so, Tim Berners-Lee is now at MIT and has it; he's in the building just a 100 yards away from where Abel and I have our offices. So, we feel that this is something you should understand about why your learning to program the web.

Video 11 – Overview Of World Wide Web, HTML, And JavaScript

So, let's position ourselves with respect to the World Wide Web. We're learning how to program the web and it was invented by Tim Berners-Lee. Also, Tim Berners-Lee is now known in the early 90s. And he invented a protocol called HTTP. So, any of the messages that we're sending around the web are usually HTTP-type messages. So, we need to understand those. Here's positioning the web with respect to other things such as Internet of Things invented in early 2000s. Smartphone, the iPhone in 2007. Big Data 2009, Machine Learning at 2012 with Geoffrey Hinton. And although the cloud computing data centers have existed since the early 2000s, Cloud Native Computing really started around 2010. It has been going on for about the last ten years.

Now, the internet is driven by messages going from one machine to another, and machines are known by their IP address. So, for example, I have machines starting 18.58 and then .1.131 and these IP addresses give us the source of a TCP packet and the destination of that packet. But with the World Wide Web, Berners-Lee invented URLs, and these are much more friendly, www.mit.edu, for example. An HTTP tells us what kind of protocol it is. And he also invented HTML layout and URIs, and as I said, HTTP. And the two main methods of HTTP are GET and POST and we'll look quickly at those. And the browser invented in 1995, the Netscape browser really is an amazing piece of equipment, and we'll talk a little bit about that.

But in this course, we're mainly concerned with this upper left-hand corner, HTML, CSS, JavaScript, and we'll talk about web APIs. But most of our messaging will be HTTP. This is how we'll communicate across machines. Now, the browser, for example, can hit multiple machines. So, it can issue, get commands to five or ten or 20. If you look at some of the pages, say, of the New York Times or The Financial Times, they'll be hitting perhaps a 100 different web servers out there and the data from those hits are composed by the browser into a single page. And so, the browser is this amazing integration engine that can bring together data from various sources and we'll be looking at exploiting that. You don't need to know too much about TCP IP. But TCP has a source IP address, and port number, and a destination IP address, and port number.

Now, the IP address is the address of the machine. But on that machine are many applications running. So, to know which application you want to talk to, each application is given a port that it listens to. So, for example, a web server usually listens on port 80. It doesn't have to, but that's the default port for web servers, it's port 80. So, if we're sending a message to a web server, we'll send it to port 80. Okay, in 1995, they needed to have something more flexible than HTML, and Brenden Eich invented JavaScript, and apparently, he did that in ten days, which is pretty

amazing. And we had the first apps in 2004. So, Google started with its Google Maps. Ian Hickson invented the HTML5, always the driver of the HTML5.

And today, this is the kind of setup that you'll be programming in. That we'll be issuing commands from a client in the browser to hit a web server. Then that web server may go out to another data server, for example, and that may hit a database. And so, we might have sensors also sending messages to the webserver that we integrate together into our webpage. So, this is the kind of environment that we're programming in. So, this is just to give you a graphical view of, we are programming up here at the top layer, the application layer, and that's where we'll issue HTTP requests. And below that, we have the TCP layer and the IP layer. And then we get down to the physical layers that actually move photons or electrical currents around the world.

And they get translated back up again to HTTP. Okay. So, the way web servers work is the browser issues an HTTP, get, for example, a request message to a server, and the server issues a response message, usually containing HTML and perhaps, JavaScript. And so, this is the pull mechanism. And the webserver doesn't remember these transactions. The message comes in, it responds to it, and then it forgets everything. But that's how it scales, it's because it doesn't have to store state. So, that's the kind of cycle we're going through. Okay, so let's take a look at how we might issue this programmatically. So, not all HTTP requests are issued from the browser. You don't just type them in.

They can probably programmatically issued as well. cURL, which you can download for your machine. I suggest you do. We're not going to use it yet, but we'll use it later. cURL issue these commands programmatically. So, for example, we could issue this command in the terminal window. Okay, let's go and do that. So, here we are in the terminal window and I'm going to issue a cURL command. I've already downloaded cURL on this machine. So, I say 'curl', and then I give it the address of the machine I want to hit. This is an AWS machine in the cloud and I have a 's3' bucket there, and I have, a bucket is called 'pollysnips' and in that, I have a file called, 'contats', and it's a JSON file. And we'll see what JSON, 'contacts'.

So, you see that it brings back a file. Let me issue and get another file. It's at the same address, but this time it's called 'users'. And actually, let me store it in a file, so I can capture the response coming back. If I say, '-o' that's our output, and I'll call it, 'test.json'. And now, I give the filename, and it's called, 'users.json'. So, you'll see that it went out and got the file. And this is very useful if you want to download large files like gigabytes, this will actually do it very well. And let's take a look at 'test.json'. Yeah, so that captured the JSON file. So, you see it's an array of objects, and these are the properties in the object. And one of the ' "courses" ' is an array itself that has other objects in the array. So, we'll learn how to navigate through these but you can see how we can get data using cURL. cURL, basically, issues a GET command. Okay, that's it for now.

Video 12 – Introduction to The Document Object Model

So, in this section, we're going to take a look at the Document Object Model. So, the browser has in memory the HTML elements that you create. So, for example, if you create in the body, you might have an h1 tag, and you might have a div. And we might have a list of things inside that div. So, we're going to show how you can get programmatic access to, for example, one of these elements like a div. And we do that either by getElementByID if we've given that element an ID. So, we might call the element my tag. And we can getElementByIDmytag, and that gets us a handle on it.

And then, we can get at its inner text or its inner HTML and modify it. And what we're going to do is we're going to use an injection technique where we're going to get a list of posts, and we're going to take those and put them into an unordered list. So, we're going to use UL to specify an unordered list. Then we're going to inject LIs, which are lists of the post items in that div. And so we can take a very long list of posts and render them quite nicely into the DOM. So, this is all about how to manipulate the DOM. And in fact, a lot of our programming that we'll see in the future is going to be exactly about that how to make things look good in the browser. And to do that, we need to manipulate the DOM.

Video 13 – Injecting JavaScript Into HTML Web Pages

So, let's take a look at HTML. Here we have an '< html >' page with the '< body >'. And inside the body we have a '< div >'. And we've seen that divs divide the page up and help us layout the page. And inside the '< div >' is an '< /h1 >' tag, 'Hello Earthling'. So, this will print out 'Hello Earthling' in bold. And we're giving this 'id=', calling it ' "tag1" ' And this is so that we can programmatically access it later. So, let's first just run that file. So, if I drag it onto the page, let's bring it up with. Then it says Hello Earthling. Okay, so let's go back now and change that. And let's put in a '< script >' tag. And in the '< script >' tag, I am going to get hold of that h1 element by its id.

So, I'm going to say 'const tag = document.getElementById' And I give it the tag name, which is '('tag1');' the id. So, I give the id '('tag1');'. And I happen to know that that has an 'innerHTML', but I can change. And I can say, ' 'Goodbye Aliens'; '. Okay. So, that's what I wanted to change to. Let's save that. Bringing this up and bringing up Finder, and drag it on there. Goodbye Aliens. Okay, so that works. Now, I just want to show you there's something you need to be careful about. Suppose I'd put the < script > tag above. Suppose I put it here. Now, do you think this is going to run? Well, you might guess that it doesn't, but let's actually save that. And now let's go back. And so, remember, Hello Earthling.

But let's take a look. I'm going to bring up the IDE because I'm pretty sure there's an error. So, there is an error on line 4. Let's bring up the source and look at line 4 here it is. It's saying tag is not defined. Let's run that again. Here we are, and let's hover over tag. It's null. Now, the reason

that it's null is if we go back and look at the code, it's because we're executing these statements in order in the browser. The browser just runs through these in the order you've given it and so, it tries to execute this 'document.getElementById'. It's looking for an element with an id of tag1. But that hasn't been defined yet. It's not until here that comes into existence.

So, we have to be careful about executing things before they come into existence on the page. Now, there's a way around this where we can tell it to wait until everything has been loaded in. The command for that is 'window.onload'. We say '= function()'. And what this is going to do, it's going to run this function once everything is loaded. So, let's save that. Now, let's execute again. And what is going to happen, is it sees this window.onload and says, This won't be executed until everything down here has loaded. Okay. So, let's go and do that. It says Goodbye Aliens So, it does run. Let's make another change.

Okay, let's change this a bit. I want to make sure that the aliens leave so I want them to get on their ship. so ' 'Goodbye Aliens: ship leaves in' ' so many seconds. And I want to put a 'time' in here. Okay. So, that looks good. 'Goodbye Aliens: ship leaves in'+ time+ 'secs';'. Now, I need to inject time. I'm going to do it like this. So, I'm going to say time. Let's make it a 'var' so it's available. I'm going to put it inside. '= 100;'. We're going to have a countdown. And we're going to say, we're going to have a function 'getTime='. We're going to make it a fat arrow function. And we going to call this inner tag in that.

Okay, so that's that. Now, I want to reset the time in here to 'time = time - 1;'. Okay. And outside here now. Let's get time. I'm going to have a timer called 'setInterval'. And I'm going to call it '(getTime,'. And I'm going to call this every half a second '500);' milliseconds. So, I'm going to call getTime here it is every 500 milliseconds. I'm going to decrement time down and that will be printed in there let's have a look of it works. The index. Goodbye Alien: ship leaves in 90 seconds. So, they better get on that ship.

Video 14 – The Document Object Model

So, when we write a webpage that has HTML and JavaScript, we need to realize that the browser is interpreting that and filling in what is called the Document Object Model, the DOM. And we start at the top level with the document, then the < html > tag, and then perhaps < head > and < title >, and then < body >. And here we have an < h1 > tag. And so, that corresponds here; we have the < body >, the blue, and now the < h1 > tag, the yellow, and Hello Earthling. So, those are some of the attributes, if you like, or the properties of that tag. Now, in JavaScript, we reach in to get a handle on these things that are in the DOM memory, the browser memory.

And so, we do document, so we're referring to this then 'getElementByID('tag1');', and we gave that tag a tag. So, we can use that now to hook onto it. So, tag now points to this, and this has properties itself, and one of them is 'innerHTML'. And now we set that to ' 'Goodbye aliens'; '. So, what happens is that Hello Earthling is overwritten, and now, Goodbye Earthling is in that slot in

browser memory. So, we can programmatically now reach into that browser memory and manipulate those elements in memory via JavaScript, and that's very powerful indeed.

Video 15 – Dynamically Inject Posts into Div

So, let's take a look at some more HTML pages. So, here I've got some data. These are posts imagine that they might be posted to a social app or something and the title, and it's got content in each one. And we want to pick these up and render them. So, let's take a look at what our page might look like. So, suppose here we have a '< div', and we give it an 'id' so we can get a handle on it in our JavaScript later. And so, we're going to populate this by our JavaScript. Now, I'm going to include the source ' "./post.js" '. So, here I've got a variable 'posts', and then it's this JSON data.

Now, notice that because I've got a 'var' in here, it's not pure JSON. This is JavaScript. If I didn't have this var, it would be JSON, and we'll do that later. But here, I'm including this JavaScript if you like,' "./post.js" '. So, we've got their posts now, and it's that array of objects. But let's make sure first, how do we inject that in here? Well, we need to do a 'document.getElementByID'. So, let me put that in there. So, 'getElementById', and I want to get that < div, so it's called '('container');'. Okay, and now, I've got, well, I need a variable here. So, 'document.getElementById', gets a handle on this in the DOM.

And now, the 'innerHTML' can be set, and now I'm going to set it to this and this 'html' here, okay? Now, later on, we're going to inject much more than this, but let's see if that works. And so, I save that. Let me grab a window here. Okay, and it's Hello World. Okay, so what I want to do now is I want to get run through that array, and instead of the html being this, I want the html to be a little bit more complicated. I got some code here that I wrote just before. So, instead of this html, Well, I've got it here so I can delete this; posts, remember, is included here.

So, 'posts' is an array. I'm going to get the length of the array. I'm going to set html to be just no characters, just an empty string. And now, I'm going to loop over these posts. I'm going to start off with 'html' being equal to, I going to want to put in a list in html. I'm going to give it a 'class="post" ', and then in ' '< h2 >' '. I want to put whatever is in the 'title', and in ' '< h3 >' ', whatever's in the 'content'. So now, html is going to be, well, a long list of these. Let's see if this works. Okay, so this, okay, pretty good.

It did what we want. It injected that list, and that's exactly what we wanted to do it. Perfect! So, we can clean this up a little bit more, and what I'm going to do is I'm going to put up all this in a 'function renderPosts', and I'm going to pass in the '(posts, container)', for that, and I need some, here we go. So, that's now a function. Okay, and now I need to call that function. So, let me put in 'renderPosts('posts,', and we'll see later why I'm doing this. And this is the 'document.getElementById('container'))'.

So, that's this, and I don't need that, and that's my function. Okay, and we'll get rid of that. Now, we just got one call to 'renderPosts'. We post in the 'posts', and the 'container', and so, now those

are being passed in, it should run. Let's just check. Load it again. It still runs. Okay, so we're doing quite well. We've seen here how we can programmatically populate this '< div'. And this is exactly what some of the modern packages do, that they will dynamically push different layouts into this single '< div' ' "container" '.

Okay, so here we're just pushing this in. But you could imagine we could push anything we like into this '< div', and it will be rendered there. So, we can dynamically change our webpages. But this is showing you now how to inject content using JavaScript. So, here we're taking data; later on, we'll search through the data and pick out different pieces. We could, for example, have just rendered the 'title' or just rendered the 'content'.

Video 16 – Render Using Fetch

So, we saw how to render a list of posts onto a web page when that list was in a local file. And we picked it up like this. We got the local file. And it was JavaScript because it had a 'var', 'posts.js' equals, and then the data. And so, we already had posts available to us in this file. It was defined. Now, what I want to do is I want to pick it up from a remote web server. In this case, we're going to hit a web server, and its URL is going to be on Amazon Web Services, so it's in the Cloud. And this time, we're going to pick up JSON. The file actually is in JSON, so we need to go and get it.

So, we had this function 'render' that would render into the 'html', into the 'container', into the 'div'. So, this is the 'div' up here, and we used the 'id' to get a handle on it here. So, this references that 'div'. And we're going to modify the 'innerHTML' to write out that list. Okay, that's what we did before. Now, to get the data from the remote URL, we need to use this function, which I've written. It's going to use 'fetch', which will fetch data from a remote URL. So, we get, we hit the webserver, and it gives us a 'response'. That response is JSON.

So, now once we've got the JSON, we need to turn it back into normal JavaScript. And this will do that. This will turn it into a 'posts' array. And then, as before, we can do 'renderPosts', which will render it up here into that 'div'. Okay, so the only difference now is we're going to hit this remote server. Okay, so let's do that. And I'm running a website at localhost:8080. So, if I go to that, it's going to automatically pick up that index.html file. So, there it is, it goes, and it hits the remote site. Now, to show you that it's hitting a remote site, let me bring up, I'm going to hit F function 12.

Now, something we need to be careful about is that often it will pick up a cached file and not the actual file. So, to refresh that, go to More Tools and then Clear Browsing Data. There we go, Clear Data. And it'll clear any cache. So, here localhost, let's squish it down again. I've 12 here. Now, let's refresh this page. Yes. Now, you see it going out here, hitting posts.json and getting it back from a remote website. And if we dig into these things, we can see exactly what it brought back. So, here post.json, this is a file that it retrieved. So, we can see the performance that this did go out and get that file back, and we rendered it.

Video 17 – Introduction to The PacMen Exercise

So, in this exercise, we're going to create PacMen on-the-fly. We're going to put a button on our webpage, and that when we click that button, a new PacMan will be created and will start moving around, bouncing off the walls. And so, we need to understand how to store all these PacMen, and we're going to do that in array. And this is typically how we store groups of things. And so, each time we create a PacMen, we're going to push it onto the end of that array.

Now, we're going to keep track of where the PacMen are in our code, and then we need to mirror that in the DOM that we're going to have to position our image in the DOM. And so, in some sense, you have a mirror in your code of what's happening in the DOM. And actually, this kind of virtual DOM, as they call it, is often used as a technique in larger codes like React. So, that's our PacMen exercise, and I hope you kind of understand the goals in that.

Video 18 – Factory for PacMen

So, here's our new game. Lots of PacMan moving around, bouncing off the walls all with different speeds that we can add new ones. So, we can keep adding to this. Okay. So, let's take a look how we build this. So, in this exercise, we're going to create a game with lots of PacMan, and they're all moving and we can add PacMan at any time. So, we can add PacMan, and then we can start the game, and then we can add more PacMan. Now, these PacMan bounce of both the vertical walls and the horizontal walls. So, we've got to check the walls for every PacMan. And every PacMan has a random velocity, to begin with, and starts at a random position. So, let's think about the design problems that we need to address.

So, first, we've got to generate lots of PacMan. So, we need to write a kind of factory that can produce them one at a time. So, when we push that button, we're going to call this factory and produce a PacMan. And then we need to think about how do we draw them on the screen. So, each PacMan needs to have a position and the velocity, and we need them to draw it on the screen. So, if you remember how do we draw an image. Well, we set, we just have an < img> tag, and we set the position, the top, and the left. So, we know how to do that, but we need to do it dynamically. So, with every PacMan, we need to make an < img > tag and add it to the DOM. So, we're going to have a < div > tag, and we're going to add to that in the DOM every image that we create.

Now, as I said, each PacMan needs to keep track of its position and velocity. So, let's use as position, an object with an x-coordinate and a y-coordinate. And for velocity, we'll create an object also with an x and a y. And that'll be its velocity. And then as I say, we set the < img > tag, we set the style.left and the style.top, just as we did before with the single PacMan moving across the screen. Okay, so I'm going to give you some clues. So, this is how I want you to make a PacMan. So, 'setToRandom', you need to write this function, but all it does is comes back with an object

that looks like this. So, we're going to return something. So, we need to calculate these values. That's all. So, velocity will be an object. With let's suppose an 'x:' velocity of, that scale it between here '0' and '10'. So, there's a function called 'Math.random'.

And it returns a number between 0 and 1, so you scale it, and then it'll return a number between 0 and 10. And for 'position', we're going to do the same, but this time scale it, so that the position is between '0' and '200' in the 'x', '0' and '200' in the 'y'. So, now we've got two objects, 'velocity' and 'position'. And you see at the end we're going to return them. So, when we make a PacMan, we're going to return the velocity and its position. It's kind of the state of that thing. Now, we've also got to take care of the < img > and the < div> tag. So, we get the < div > tag. I called it '('game')' here. And we'll see with the < img > tag in a minute. But we need to get hold of it. Now, I am going to create a new < img > tag, and 'document.createElement does that for you. I'm going to set the 'style.position'. It has to be ' 'absolute' '. And I'm going to set the image.

So if you want to have different images, this is where you'd put them. So, each time you make a PacMan, you might want to make a different image. And you might want to make them different sizes. Here, I'm making the 'width' a '100;', but that's arbitrary. And then we set the position. So, now we've got this tag, and now, we append it to the 'game'. That's the < div > tag up here. So, now we've got a new < img > tag appended to the < div > tag. And we can make hundreds of these PacMan like that. So, now we've made one, and we return this. And we need to store it somewhere. So, we're going to need an array that we're going to put this in. So, we'll see that. Now, to update position, we're going to loop over this array for 'pacMen'.

And for each of them, we're going to check whether, now, here we're going to check its updated position. So, here it's updated position. But I'm going to check that in collision. So, I'm going to pre, I'm going to look ahead and see if I'm going to hit the wall in the next time step. If I am, I'm going to set my, I'm going to bounce back immediately. And so, the 'velocity' will come back as maybe a minus velocity, out of this 'checkCollisions'. Okay. It's going to update this item, so that's the pacMen. It is going to update its position, and it's going to update in the DOM, its position. And this 'setTimeout' calls 'update'. Once update is called, it's going to call it every '20' milliseconds. So, it's going to call it quite quickly.

Now, to 'checkCollisions', you can go through this, but all it's doing is checking. I'm adding the 'velocity' step here. So, that's my increment, if you like, of how much the position is going to change. And so, I'm looking ahead and I need to add the width of the image because remember we're keeping track. The position is the left-hand side of the x-coordinate. And it's the top in the y-coordinate. So, check that this is correct. Now, let's go to the next one. And here we see, here the '< div >'s. So, we've got a ' 'game' ' div. And then we've got two 'button's. One button makes a PacMan. So, I just showed you 'makeOne( )' here. And remember I showed you 'makePac( )'. So, I'm calling 'makePac( )', and it returns, remember what it returns? It returns an object.

So, remember what it returns? It returns an object with a velocity and a position, and also a pointer to that < img > tag. Okay, so it's got them all. And now, we push it, so, into this array. So, push kind of pushes things down. Imagine it's like you've got a stack of plates and you're putting another

plate on the top, and you push it down. So, we're just adding to this array of 'pacMen'. And then we start the game by clicking on 'Start Game'. We call ' 'update' ', and ' 'update' ', remember, it's going to call itself continuously. Okay. So, there's one function you've got to write, and that's that random function. Remember? We go back here, 'setToRandom'. And in that you call a function called 'Math' with capital M '.random' with a '( )' after it.

And that will return a number to you between 0 and 1, and then scale it. So, I'm just leaving you a little work to do. Okay and that will give us, hopefully. Let's just go there. So, here we are with the game. We can Add PacMan. And now, we can start them, and we can add more. That was a slow one, there is a faster one. And we can keep adding them So, once you set this up, it's surprising how well it scales, that we can create hundreds of these. And these are all images that moving. And if you're clever, we could change the size of the image. It could be like a tennis ball getting compressed when we hit the wall. We could squish the image up and make it smaller. And then as it leaves the wall, make it expand again.

Now, there's a few things that are new in the way I'm coding here. So, we're keeping track of when we make a PacMan, we're keeping track of its 'position' and 'velocity' here. And we're also keeping its position in the DOM. So, in some sense, we've created a, what you might call a shadow DOM. This position and velocity, you could imagine we could separate these two pieces. This top piece is creating 'position' and 'velocity' in the shadow DOM. And then this is updating it in the real DOM. And you'll see if you do more computing in generating websites that a lot of the time, they'll create what they call a shadow DOM.

And so, we're kind of doing it here. Now, the next thing you'll see is we're creating here an object, but we're not putting colons. But the default is if it sees this, it constructs an object with 'position:'. So, the 'position' is the key, and then ':' the value is the value of 'position'. And the same for the 'velocity', 'velocity: velocity,' 'newimg: newimg'. So, this is a shorthand. And here this is the 'Timeout' calling, 'update( )' is calling itself. So, that's another new thing. Also here, we've got a 'forEach' statement. And 'forEach' operates on an array, and there's a callback. And we'll be dealing with callbacks extensively in the rest of this course.

Almost all of modern computing makes use of callbacks. And let's just take a look at this last one here. What are we doing here? Well, we're calling 'makePac( )', and remember it returns that object. And we're pushing that object into the pacMen array. So, the order that this is executed, 'makePac( )' is executed. It returns something. Whatever is returned is pushed into the array, 'pacMen'. And you'll see 'pacMen.push'. And if we wanted to take one away, destroy one or do something we could do 'pacMen.pop'. So, push and pop do the opposite. One pushes the plate on top of the stack, and pop takes the plate off the top of the stack.