ROBERT GREEN: Okeydoke. So we're getting a few more chairs if you want to-- I don't know. If you're like me, you kind of prefer to stand up. Yeah, OK. So welcome. Hope AutoCAD-- or Autodesk University 2016 has been fun for you so far. My name's Robert Green, and most people over the years have come to know me from the CAD Managers' newsletter, things for Catalyst Magazine. Anybody a member of the CAD Mangers' Facebook group? I know I talked to Corey. A few people? Cool. Nice to see you.

Let's just go ahead and kind of outline a little quick bio and what we want to do in the course. And that way if this isn't what you expected, you can go someplace else. Basically, I am a recovering mechanical engineer. I was the guy who is dumb enough to raise his hand and say, yeah, I want to be the CAD manager. And the rest is history.

So I kind of, through no great fault of my own, wound up becoming unemployed in 1991 and have stayed that way. So I do this kind of stuff on a contract basis. And I tend to emphasize CAD standards, CAD management stuff in what I do. And therefore, I find that programming techniques are a very excellent way to get standards in place and in force.

So how many of you are the CAD manager, just out of curiosity? You actually have the title? Eh? How many of you are really, really, really comfortable with AutoLISP already? Really, really, really. How many of you are kind of like eh? All right. OK, cool.

This is my 21st Autodesk University, by the way. I've got a Facebook group. I've got a Twitter handle there. And I've got a website for you. I promise I won't send you any junk. Be happy to have you join up in any way or communicate with me in any way you might see fit. Please feel free.

Here's the philosophy of the course. Here's kind of how I see this. I see that the projector is truncating a little bit of the screen, so hopefully we won't have too many problems with that. My basic feeling is, well, why AutoLISP? Why would I want to know this? Why would I want to get more in-depth with it? Isn't it some kind of dead, obscure language like Latin or something like that?

That's a reasonable question. It is profoundly-- a profoundly powerful tool for managing AutoCAD platform tools. Not just AutoCAD, but AutoCAD mechanical, architectural, C3D. So anything that's running off AutoCAD as its core platform can be automated, controlled, managed via AutoLISP, and that is the reason that CAD managers should know it. Now, that doesn't mean that you have to know every little bit about it. It doesn't mean you have to be at a developer level. It just means that you have to know a certain subset of the functions and become acquainted with it. And if you know that much, then you can be very effective in managing your AutoCAD installations.

And by the way, from a CAD management point of view, I'm not talking about just one machine, I'm talking about everybody. So all your users around the network environment. I hope I don't have to get in a yelling match with next door. We'll find out.

So the way I'm going to dissect the course is I'm going to kind of start at the beginning-- I'm not going to assume that anybody knows anything. I'm just going to start off and define everything that I'm going to need and the functions I'm going to show you. I'll explain and backfill a little bit of theory as we go along. We'll start slow. We'll speed up. We'll go from lower complexity to higher complexity. And I've kind of found as I've taught the course over the years that I'll have people say, I've been doing LISP for a while, but I never knew that.

So my hope is I can deliver a few little nuggets of wisdom as we go along and kind of build everybody's skill set as we do so. And by the way, all the examples that I will be using are in the course guide. You do not have to struggle to write this stuff down. It's already written down for you. Just try to watch, absorb, and learn. And then you can go back and look at the examples that are written down at your own convenience later when we get back. Makes sense? If that's not what everybody wants to see, then you can go grab another class right now. I won't feel bad. Three, two, one. OK, here we go.

We're going to define a few key variables and functions and a few things that, in my experience, a lot of people who have even done LISP before don't necessarily know all this stuff. First of all, AutoLISP. Anybody know where LISP came from? It was the first structured programming language, and it was used in the Korean War to calculate the trajectory of ballistics shells that were flying off of battleships. So if you ever play AutoCAD fun facts, or you ever get into that at the bar, and there's AutoLISP trivia, now you know.

So it's actually one of the oldest structured programming languages on the planet. A lot of people don't know that. It has a file suffix type of .LSP. There are some LSP files that are very, very important to the way that AutoCAD tends to run as it runs in an ongoing basis and as it starts up. So let's make sure that we understand what those files are, and we understand what to be careful of.

If you were to make your way into the support filing structure of your AutoCAD-- and of course, based on your AutoCAD application, you could have many support directories. And depending on vertical products, those can exist in many multiple places. But as you make your way down into the basic support category of AutoCAD, you're going to start finding some files that look like this. ACAD20XX where that is the version. So 16, 14, 12, 08, whatever version-- and by the way, I see a bunch of 2008 out in the field still. So it's still out there.

There can be an ACAD.LSP file, but that is a file that you create and control. There will be another one that is called ACAD2000 version doc.LSP. That's a system level file. And then there will be an ACADDOC LSP file. And again, that's something that you create. So please notice the symmetry in the naming structure. ACAD2016.LPS loads as AutoCAD starts. But ACAD2016DOC.LSP loads every time a new drawing session is initiated.

See the difference? Because you might start out once in the morning, but you may be in and out of 45 files throughout the course of the day. So the loading behavior is in fact different. We're going to exploit that here in a moment.

There is another type of file, a .MNL, which is actually LISP. If you look inside of it, it is LISP code, but it must be named exactly the same way as your CUI file. So if you have an electrical.CUIX, then you can have electrical.MNL. And that AutoLISP code will load only as the CUI file loads. And that can sometimes be a useful trick to load in code for discipline specific CUIs or workspaces. CAD managers are into that, so that's why I mention it. The reason that I go through all this is to state the following. These live in your support folder structure, and they load in in the order in which I have presented them.

So let's do a little Q&A session just to make sure everybody is awake, because they always give me programming classes right after lunch break. Go figure. And then they wedge us all into a nice warm room, and yeah. That's what happens.

So if you go into the support folder, and you locate your ACAD2016.LSP dot LSP file for sake of argument, and you double click on it, you're going to see it pop open in Notepad, because that's the default file association that AutoCAD uses. Now, it'll come up, and you'll scroll through page after page after page of stuff that makes absolutely no sense. You might as well be reading Sanskrit. This is something that is used by AutoCAD for its internal purposes to define functions, backwards compatibility to prior releases, etc.

Here's the question. Is this a good file for you to screw around with? Not hardly. So if you click open a file, and you see a bunch of stuff come up that you didn't write, my recommendation is close it. Don't mess with it. Same thing with the ACAD2016DOC.LSP. And another thing is if you have any third party types of software tools, a lot of times they'll put little function calls in these files so that they load incorrectly, and you certainly don't want to mess that stuff up.

What we do have, though, is this ACADDOC.LSP file. That's yours, and it will open every single time a new drawing session is created or is open for editing. So this is a great place for you to put your custom code. I would argue it's the best place to put your custom code. Are there are a number of other ways you could proceed? Yes. This is the way that I would recommend though. So just to kind of state that.

You're going to create an ACADDOC.LSP file. You're going to load it in a support folder, wherever that may be. As you make changes to the file, I know you will keep it religiously backed up, yes? So then when you mess up, and everything is hideously screwed up and won't load, you just simply delete it, restart your AutoCAD, try again. That's not so bad. There's only so far you can get with that.

What I will do then is I will tend to keep this file on my local machine as I create it, edit it, debug it. And then once I'm convinced that it runs, and I've pounded on it for some period of time, I will then move it up to the network so everyone else can leverage it. You really don't want to make live edits to something that's out on the network. And run into a situation-- I like to describe it as changing the tire on a moving car. It's not easy to do.

So let me just spin through that real quick. Here you basically just go-- oh, and by the way, I'll show you a variety of different versions and some of the screen grabs in the handout up here. This methodology really has not changed. I really don't care what version you're in, this same exact technique will work. So you'll see a bunch of different screen grabs as we go through.

So what you'll do is you'll go into your current profile. You'll go ahead and look in the Files tab. You'll find out where your support folders are. And what I would like you to do before you get any further is create yourself a new testing profile just to isolate anything that you're going to do or edit. From there, it'll be very, very easy to switch back to your vanilla profile or whatever your vertical profile is. We'll just simply isolate your testing environment a little bit and insulate you from damage.

So now what I would do is I would go ahead and create a new folder, which would contain my code that I'm going to use for testing. That can be on a network drive. It could be on a local C folder. I don't care. Whatever makes sense for you. You create that new folder. It shows up at the bottom. I now just bump it up the list until it gets to the top of my support folders, and that's what AutoCAD is going to load first. Easy.

Now, in AutoCAD 2014 and later versions, a new thing has come into play that's called the trusted locations folders. And this is in response to a pretty nasty little virus that went around years-- anybody get zapped with that ACADLISP virus? Self-replicating little thing. So what you have to do now is go into the trusted locations folder, and you have to add whatever folder you added here. You got to make sure to make that a trusted location. That way AutoCAD, if it is attempting to load the LSP file, it will only do so if it's in a known, trusted spot.

And if you've ever tried to chase down that virus in years past, you'd know why they did this. It's annoying, but it's a good thing. Just to make you aware that you have to do that.

Here's the first thing I always put in my ACADDOC file. I put a prompt in there that just basically says the ACADDOC.LSP file has been loaded. That way every time you start a new file or edit something, you'll see that spit out at the command line, and that's a good indication that your code actually made it. So basic, but worth mentioning.

Just a couple of things, rookie mistakes that I see made all the time. When you save this file, make absolutely sure that it is saved as a ACADDOC.LSP. A lot of times, people will go in and create a file, and they won't notice that down here it says file type is .TXT. That will create an ACADDOC.LSP.TXT, which will not load. Enough said.

Having said this, there is another way to do this. And it's called the Startup Suite, which is fine. I will tell you that what I'm trying to move towards later is to have this load in and automate-- totally automated, so that it's independent of any other files, and ACADDOC.LSP gives me a cleaner way to do that than the Startup Suite. It's valid, but I'd recommend the other.

OK. So now that we know what some key files are, and we know where they live, and we know how to get started, let's start looking at some programming code that we might throw in there that would have some good context for a CAD manager. AutoCAD functions by tapping into and constantly reading and writing from a collection of what are called system variables. Now, if you've been using AutoCAD for a really long time, you used to actually set your dimensioning variables by going to the command line and saying, dim var. And then you had to know that it was dim dli was what you wanted to change. So you used to have to have a pretty decent knowledge of these things-- these AutoCAD.

Now, these days, that's not true. You're pretty much insulated by this graphical interface, and you don't really know what variables are being affected. So more on that in a moment.

AutoLISP gives you a way to set any system variable using this type of syntax. So here, I'm setting a fillet radius to 0.25 units. So the next time you invoke the fillet command, what's the radius going to be? 0.25 in whatever prevailing unit system you happen to be in. You never had to go into the fillet command. You never had to type anything in manually. It's just-- there it is.

Here's another example. The expert system variable. This one's great. How many of you guys know about this? If you're hacking around on LISP, you probably know. The expert system variable, when set to the maximum value of 5, basically suppresses all of the idiot prompts like the "are you sure?" prompts. Which is kind of neat when you're writing a program, because it's hard when you're writing a program to understand every place that an idiot prompt can pop up. So that's one of the things that I tend to kind of put in mine.

Here is a way that you could set the current dimension style, for example. So the next time somebody goes into the dimensioning routine, what kind of dimension will they get? Again, an AutoCAD mechanical ansi style, assuming that they're in AutoCAD mechanical.

Now, if we put the word GETVAR where I have SETVAR, it will get values from the stack rather than setting them. And this will be a technique that we will use to pull variables out, manipulate them, and then push them back in an edited state. So you need to know both sides of it. Just note the syntax, right?

If you've never done AutoLISP, a left paraenthesis-- you've got to be balanced by a right one. That's a good rule. Any quote marks must be in balanced sets. And you'll notice that there is a significant space between what are called arguments to the list. So those are significant, and they do need to be there.

Command line access. This is another thing that I'll use a lot the routines I'm going to be showing you. So here, we are simply opening with a left paren. We're simply opening an input channel to the AutoLISP interpreter, and we are saying, I would like to issue a command. So this is functionally equivalent to typing into the command prompt. So what I now say is, in quotes, "viewres," which sets the view resolution for things like how smooth curves are, how smooth polylines appear when they're zoomed in on. So if you sat at the command line, and said, "viewres," it would said, would you like fast zooms, yes or no? What have I responded? Y is for yes. And it would then say, what value would you like from a value of 1 to 20,000? And I have replied 5,000. So if it's inside the quotes, it's just like I typed it, correct? That's it, by definition. It's just as if you typed it.

Now, what if I typed in something that made no sense like viewres, yes. And it said, what value would you like from 1 to 20,000? If I typed in bozo, what would happen? The command is going to spit back. That's an invalid response or an invalid value. So the burden is on whom to make sure that this is correct? Us, whoever the programmer is.

How would I make sure that the current color is set to a bylayer state? Here's an answer for that. What's different about the color command? Dash. Which does-- it forces the color command down to the command prompt as opposed to being a dialog box. And LISP is a-- it is a command line driven interface.

Here's the same thing I would do with line type, but I want to point one thing out in that example. If I go to the command line, and I say, -lineype, it will come back, would you like to create, set, load, or question mark? I have said set. It would say, what would you like to set it to? I have replied BYLAYER. And then it would come back, and it would say again, create, load, set, or question mark?

So why have I put two quote marks with nothing inside them? It acts as an enter. So you always have to think about how do I logically get out of the command. And by the way, how do I know that that's the prompting sequence? I went in and typed it in manually and jotted it down on a piece of paper just like you guys are going to do it. I'm not psychic. But you are going to get to know AutoCAD from its command line perspective a lot better.

Here's a way that I could load a CUI file. Now, an interesting thing pops up here. If I put this code in my ACADDOC.LSP file, which loads how often? Every single time AutoCAD starts, every single time a file is created or opened. What's going to happen to this user every single time they start an AutoCAD session? They are going to get whatever color and line type behavior-- and yes, I'm talking about MicroStation users. People who laugh in the civil environment know exactly what I'm talking about. They're going to get the right CUI file in memory.

So if I had a chronic, nonstandard abuser, every single time they start AutoCAD, what's going to happen to them? They're going to get reset. And this is-- my favorite expression of this is this a war of attrition. And I'm going to kill them, because I'm just going to keep doing it. I'm just going to keep resetting them to standard, and eventually they'll quit fighting it. So this is really a great way to get everything set up right.

Now, does it matter what profile this user is in? No, it just happens, which is beautiful. It's a non-- it is a profile agnostic standards enforcement tool. Even if you know no more AutoLISP than this, it's an extremely valid CAD management tool.

One more thing we need before we dive into some of the functions. There is a function called SETQ, which means set equal to. And this is a way that you can create named variables in space-- they stay in memory space-- available to our program for later use. And we will see a number of cases where I do this. So a very simple way would be to establish a variable name, and just put my name in there. So SETQ, the variable name-- my name-- then becomes this string. That's all there is to it.

Well, that's not tremendously useful, though. Wouldn't it be cooler if you could establish a variable that knew who was logged in and kept track of it? Here's how we could do that. So I introduce one other concept here. Well, two actually. There's GETVAR, which I mentioned earlier. So I go out to the environment stack, and I say, get the variable called the loginname. Name And it gets it, and it then sets it equal to a variable that define here.

How many sets of parentheses? Two. Which one evaluates first? The innermost. So the first thing that occurs is what? Gets the log in name. The second thing is-- it stores that in that variable. And you're going to see me use that kind of trick quite a lot in some of the routines I'm going to show you.

Another thing that's very useful is to set variables for things like paths. It's real nice to just have a variable driven path. And then any time you want to use a path in any of your other routines, you just use that variable name. Because now, if you want to change all the paths in 100 different programs, how many places do you change it? One place. Very useful. The takeaway to note here is double backslashes. Do you see that? Why? Because it is. There actually is a reason, but it's kind of obscure.

OK. So what we're now going to do is we're just going to take the basics that we saw, and we're going to start putting them into some functions. And this is something that CAD managers can do to achieve a lot of keystroke automation. So everybody should be familiar with this. So let's just kind of dive in and kind of look at this. The thing that I really like about this is it allows me to shave keystrokes off common functions. Users like it, because it's easier for them to complete a task. I like it, because if I automate a task, I can put my standards in it. So everybody see how I came to that conclusion?

So if I want somebody to do something in a consistent way, I find the most compelling way to do that is to simply automate it. The coolest standard is the one they don't know they're using. Let's look at an example.

So the only thing that I've introduced here is the concept of defining a function or defun. So define function, C:, which means a command function called ZA. Apologies to any Europeans or Canadians. That would be zed. But ZA down here. So I'm going to define a function called ZA. This will be, for all intents and purposes, a new AutoCAD command. That is not totally accurate, but it will behave that way.

So if you were now to go in the command line and type ZA, what would happen based on what you're seeing on the screen? Yep. It would call a command prompt. It would say zoom. Zoom would say, well, what kind? A is for all, W is for window, P is for previous. And I replied A is for all. So as long as I get legal syntax in there, it'll work.

No, you don't have to enter. You don't have to enter. Because he would say command, zoom, and as soon as you say A is for all, the quote marks basically are functioning like the inner. Yeah. The line type one is a little different, because there was an extra prompt in it. Yeah. Now let's just kind of go over some of the overhead here, because these are quite-- I could just say do it like this, and don't ask questions. But there's going to be some things people will ask.

So this parenthesis here balances that one. I think of it as the cover-- the front and back cover on a book. Define function, C:, command form, ZA, what you will type in, and then this left and right, which is the local variable input list, which is not valid here. I'm not using that. I don't need it, but there does have to be a nil list defined there. So trust me. I hate to do that, but trust me.

Then here, princ, which stands for print console, basically flushes the memory out as the function closes-- as ZA closes, it cleans the buffer out logically and terminates the function. If you do not put this here, what will happen is it will work, but you're going to see the word nil spill out several times at the end of the function. And your users are going to call you and say, what the hell is this nil thing? And the best phone call is what? The one you don't get.

So what we'll do is just make sure to put that in there, and that'll suppress it. So basically, every function you're going to write is going to have a leading line like that. And it's going to have these two at the end like that. Your code goes up in here. So that's a quick, dirty explanation of why you want to do things that way.

You simply type that code into your ACADDOC.LSP file, save it, and the next time you start a drawing or AutoCAD, it automatically loads and invokes. Did you notice that I put my code in front of the status prompt? Why did I put my code up here instead of below there? If something bombs out here, I'll never get that prompt, will I? So that's a pretty good indicator that all my code ran if I see this prompt in the command window. It's a pretty good indication I made it that far.

Writing this stuff isn't hard. It's figuring out why it doesn't work that's hard. Here's a few more. The Auto Purge function. There's a great story behind this, which-- but not just at the moment. I'll tell you that shortly. This is a way that you could create a function called atp. And it would call a command line -purge. Why is it dash? Command prompt. It would say -purge, and then it would say, well, what would you like to purge? A is for all.

It would then ask you for the naming mask, which I have replied with a star, which means all. It's like a wild card. It would then say, would you like to review every item as it is purged? To which I have replied no. And then at the end of the purge operation, I do a quick save, or a qsave, just to make sure the file's been saved properly in its purge state, because we all know users that purge, but then they don't do what? They don't save it, so what's the point?

So this was actually in response to a CAD manager who was complaining that-- people never purged their drawings. So we went ahead and put this out there. And you said, OK, all I got to do now is type in atp, and you're good. To make it really easy to purge stuff. So we ran that for a few days, and he said, well, I'm getting complaints, because people are purging out line types or things that they need. Really the blocks are the problem. Just purging out the embedded blocks are the problem.

So we edited it a little bit. And you see how I changed a to b? So this is now purging only blocks. Purge what? Purge b for blocks. Works great, right? But users do what? They don't use it. So the files still aren't getting purged. Hold that thought. Do something a little bit fun with that.

Here's a couple more examples. I could define a VR function to set viewres quickly and easily. I could do a BL, a BYLAYER, keystroke that would set color and line type characteristics very easily. So you can see how I'm just crunching multiple steps, multiple keystroke intensive steps, down into something that's one or two keystrokes that's very easy for the user to use.

And it's funny, because I can always tell when somebody comes from a place with a good CAD manager, because they'll say, where's the ZVR command? And we got-- AutoCAD doesn't have a ZVR command. Well we had one at the company I worked at. So you can tell their CAD manager does this kind of stuff. It's really not complicated. It's almost kind of cookbook. The stuff you just need to be aware of is what goes in on the command lines. Make sense? All right. Let's ramp up the complexity a little bit.

And let's do an example with a fillet. So way back once upon a time, AutoCAD didn't have a term command. I don't know if anybody goes back that far. And it's true. It really didn't. But what it did have was a fillet command. And if you set the fillet radius value to zero, it fillets to sharp corners, which kind of works like a trim. So that's kind of a cool example to look at.

So what I've done here is I have defined a command called FZ. And as soon as the user types in FZ, I set the fillet radius variable to zero. I want you to notice 0.0. Did you notice in a prior example I said 0.25? Always put something in front of the dot. Always, always, always, otherwise you're going to get something that's called an invalid dotted pair. Those things are almost-- well, they're just not fun to track down. So always a leading zero in front of the dot.

I then call the fillet command, which is going to require user input, because I'm not psychic. I don't know what they want to fillet. So what I have is a capability in LISP that's called pause. And that will simply wait two times for the user to click on valid inputs for the fillet command. The fillet will then occur. The princ function will be issued, which will clear out the memory, and the function will close. All there is to it.

What have I not done? I took a system variable, filletrad, and I set it to something. And for good technique, I really should put it back the way I found it when the function concludes. Makes sense? Which I haven't done here. So let's beef that up and see how that will look. Excuse me. Really is a desert out here.

Now, when the user types FZ, the very first thing I will do is I will go get the variable called filletrad. See how that first line has two sets of parents, right? We evaluate the innermost one first. So I go get the current filletrad, and I set it equal to a variable called old_filletrad. Why did I name it that? Because it's representative of what it's containing. There's no rule. Doesn't have to be upper or lower case. These are questions I usually get.

The function now looks just as it did before, but after the fillet occurs, the fillet radius is now set back to what value? The same one that I stored up here. See how that agrees? So any program you use that is going to manipulate or use SETVAR, you need to take care to find out what that variable is first, store it in an old version, and then when your program is complete, you reset that using the old variable that you stored above.

So you can use this kind of like a cookie cutter recipe kind of format without getting bogged down in a whole lot of understanding what the variable names are. That's pretty decent little routine. There's still a weakness in it, which we're going to look at when we get down the road just a little bit. Well, I'll go ahead and ask. Does anybody see a problem?

OK. I heard kind of it tangentially. The great thing about this program is that it allows you to custom create something that will then prompt for the user's input. The terrible thing about this program is it allows user input. What can users do?

I don't know. They could respond in any possible way. I tell you one thing they could absolutely do is they could get to the point where the fillet is going to execute, and they could hit the Escape key. And if they did that, what would happen? Yeah. The fillet radius would never get reset, because we never made it that far. Or they could pick-- they could pick a line segment on the first one, and then they could pick a circle on the second one, which would error the command out. So there's a million ways that the thing could be entered incorrectly.

So we're going to look at that momentarily. I wanted to give you just a couple more things that you can do inside of functions, and then we'll start looking at error handlers and run back to our fillet example and finish that one and get that as beefed up as possible.

LISP gives you a couple of really pretty cool ways to interface to the outside world or call other applications. Here's one. If you open a command line in an AutoLISP interpreter or in your program, and you simply put in the keyword "browser," AutoCAD will go out to whatever your registered browser is-- I don't care whether it's Internet Explorer, whether it's Chrome, whatever-- your system knows what the registered browser is. And you can then say in that browser, I want you to open this file.

Do you see a way that you could use this for help systems? Yeah. It's a slam dunk for that, right? You can write your help files in Word, save them out as HTML, and then just pop them open like this. Make some help-- make some little macros, put them on a toolbar or something like that, and you've got your own custom help system that fires up in a browser. Very, very easy to do.

What about if you wanted to call an external program name? Startapp is the key word for that. And you know what? I should have put double backslashes in that. That's my mistake. Maybe I should fix that. I'll fix it when I reload the program notes.

So here-- I actually did get it right on the next slides. What do you know? So here's how that would alert. Here's how something like that would look inside functions. I could define a little command function called NP that would fire up Notepad. That's how that would look. I could start a custom program, any.exe. File anything that you have anywhere. So if you guys like to write-- if you've got a little .net apps or old VB6 applications that you've got that you want to call from inside AutoCAD, it becomes pretty straightforward to do this.

By the way, any of these functions-- like myprog or np or fz or anything like that-- could you start those from toolbar buttons? Ribbon elements? Yeah. Just if you're a customizer, it just becomes a valid AutoCAD command. That's really all there is to it. So we've got a pretty cool way to add these functions and deploy them using whatever kind of technique you like to use, including browser interfaces. So that one's powerful. I use the daylights out of that. And most people don't know that that exists.

All right. Let's look at error handlers. And we're going to jump back to our fillet program that we had a moment ago. Yeah. The problem here is that if the user hits the Escape key or errors out somewhere in the execution of the command, we're never going to make it down here to where we put the fillet radius back to the way it should have been. What we need is an error handling mechanism that will react should the program abort or should the user hit Escape.

There's a million reasons this could occur, so what we want to do is have a generic way that will handle it. This-- has anybody ever written their own error handlers? This is not a light bulb on thing. This is a dimmer switch coming up gradually. So I'm going to walk you through it step by step, and we may have to go back, but that's OK. This is not that hard. I promise. You just have to see it before it makes sense.

What I will do is create my own error handling routine. Error handling routines are always written-- or customarily they're written-- starting and ending with asterisks. That's because they're really easy to find as you scan through code. So is it absolutely required? No, but I highly recommend you stay with that naming criteria. Now, AutoCAD has a standard error handler. It's called just *error*. It's in the system. You don't have to do anything to get. It's just there.

It doesn't do much. Basically, all it does is it just spits out a message like invalid input, something like that. It's not terribly descriptive. What I'm doing here is writing my own fz error. This will be the error handler that will occur when I'm running my fillet command that we created before. You'll see that I put some code down here that says if old_filletrad setvar filletrad to old_filletrad. And we'll see the code for this in a second. And then there's a second line that says if there is an old error handler, we'll reset the standard error handler.

Now, this has to be here. You have to write this for any function you're going to create. Don't really ask a whole lot about why at this moment. Just use this as a cookie cutter approach. And get to the next screen, you'll start to see what's going on. By the way, this is in the handout, so you don't have to worry about typing it or writing it down.

So here's what happens. Now when I go into my fz function, I am going to store not only the old fillet radius, but I'm going to store the old error handler. So I'm going to remember what error handler was in place when I defined my function. I'm then going to redefine the default error handler to my new fz error, and that's the one that I wrote back here. I will then perform my filleting operations. And on the way out, I'll put the error handler back to the way that it was.

Does anybody see the weakness in this? And I know I'm kind of mentally blowing you away with this at the moment. That's OK. The problem is the user could still hit the error-- they could still hit Escape right about in here, and even the error handler wouldn't get reset properly, would it? So let's go back a screen.

This error handler will now not only put the fillet radius back to the way that it should be, it's putting the error handler back to the way that it should be. So no matter how they managed to abort their way out of this filleting command, everything is going to get put back the way it should get put back. You spin forward.

So this line right here is actually what protects us from the user bailing out right here. Now, I want to say this is page five or six of the handout. So it's all presented to you there on one continuous page. And you can kind of see it all on one screen, which makes it a little easier mentally to go back and forth between the two screens. But you can use this as a cookie cutter. If you write a routine called fz, create an error handler called fz_error. And just replace the fz with whatever your function is called, and just use this like a template.

The only thing that changes is the system variable that's right in here. See that? That system variable has to jive with whatever's inside your program. So that's where the two must agree. Question?

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: Yes. Yep. I usually always put the code for the error handler right before the code for the function. That way they're together, and they're easy to read. I know that's-- like I said, it's not light bulb on, it's kind of like a dimmer thing. You have to kind of look at it a few times before it totally makes sense. But once you get used to doing this, you'll have very well-behaved functions, and it won't matter how your user mangles it or hits the Escape key. You're always going to get the system put back right the way it was before the function loaded in.

So my recommendation is if you're going to create a simple function like fz or atp or whatever, you should probably create an error handler for it. It's not that hard, and once you get used to it, it's just a few characters. It's not that bad. And this will make sure that your system always gets restored correctly, and that your user basically can't scuttle your system.

The routines that you're creating now are well-behaved. They'll auto restore. Everything's good. Having said this, you could go back, and you could write functions like this all day long and never write an error handler the rest of your life, and it would still probably work 95% of the time. So if you want to ignore the error handling thing, you can. I'd rather you learned the rest of it, and come back and learned the error handling stuff last. I just want to throw it out there, because it's a very valuable technique to use.

All right. Let's move forward. We're going to continue talking about functions, but we're going to start fine tuning it. At this point, I assume that you've written a few of these functions like fz or auto purge or some of the other things that we looked at. What I now want to do is give you tools to control AutoCAD's command vocabulary. If you think about it, I've already showed you how to add a command to AutoCAD. I showed you how to add fz. Atp wasn't there before, but it works now. So I've kind of showed you how to add commands to its vocabulary.

What about taking them away? Now, when AutoCAD release 10 came out, that was a year into my first CAD management job-- real CAD management job. And we updated to release 10, and it had all these 3D primitive things in it like cones and sphere and toruses and stuff. And I'd walk past all the guys desks, and I'd see them playing with these things. And they're complaining that they can't get their stuff done. I said, well, you clearly got enough time to play around with a TORUS command. Because I can see it, so-- how about we just turn it off? Which is exactly what I'm doing here.

Let's say you're in a civil engineering environment, and the commandment is ye shalt not line, ye shalt p line. So I can turn the line command off. See how I did that? I'm tired of seeing Jason ever there piddling around with a torus command, so I'll just turn it off. And Jason's not going to come ask me, where'd the torus command go? He just isn't.

So what I've now got is an effective way to delete things-- that's not accurate. It appears as if I have deleted a command from AutoCAD. As far as your user is concerned, it's gone. So this is a really cool way to enforce standards and keep people from working around and using commands they ought not be using. Real straightforward to do. You just, in your ACADDOC.LSP, you just put in a line just like that. Gone. Yes?

AUDIENCE: What happens when a user then tries to initiate that command? Do they see something that says [INAUDIBLE]?

ROBERT GREEN: This is a good question. So the user now tries to say L is for a line. Unknown command. They press the line toolbar button. Unknown command. Just doesn't work. Yeah. There is a trick, and I am going to get to that. And I'm shocked nobody's asked me what the dot is yet. But we'll get to that in just a moment.

The dot character says that any command that has been undefined-- LINE-- that LINE command is not going to work. L is an alias for LINE, so the alias will not work. There's a macro that's associated with a toolbar button that calls the LINE command. It will not work, because the LINE command is no longer available. However, the .LINE command always remains available. But that stays in Vegas, correct?

Notice that a lot of times in my programs I tend to put dots in front of things. And that is because I do not know what has been undefined. By putting a dot in front of it, I know it'll work. So that's why you'll tend to see me put dots in my programs. This is starting to get fun now, isn't it? Yeah. Once you get past the first 45 minutes of the dry theoretical stuff, you start to see what's going to happen.

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: I love it, man. Checks in the mail, because that's exactly where we're going next. He's saying-- now, the thing is if-- well, let's go ahead. You can bring a command back from the dead by redefining it. Here's an example here. So it can sometimes be useful in a command to temporarily undefine something. Perform some functions, and then put it back to where it is available once you're done with the command. That's-- Yeah. Right. So there's definitely times when I will temporarily undefine things, but then I want to make sure I put it back the way I found it on the way out. So this is how I can do that.

Now, the problem is if your users start to understand that things can be redefined, then it's like almost a contest at this point, isn't it? They're going to try to go in and redefine things behind you. So the savvy CAD manager will always undefine REDEFINE. There we go.

Well, the thing is if somebody knows the dot-- if they know the dot, they can get around it. However, I'm, again, going back to that war of attrition thing. If they always have to stop, and they have to go to the command line, and they always have to put dot in front of it, they will get so tired of doing that that they'll just start using the standard.

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: Yeah. However, if I control the master file, and it's in a read only folder, I'm still going to win. Undefining plus functions. There is one little cool little thing that I'll just put out here real quick just for completeness. This is a way that you can send an alert from any program you write out to the user. All they can do is acknowledge it by hitting OK. But just for completeness, I want to put that there. And I'll show you an example of that in a moment.

So what am I doing here? I can tell when people get it, because that's when the laughing starts. First, I am doing what? I'm undefining the torus command. Second, since the TORUS command is no longer in AutoCAD's command vocabulary, I can define my own C:TORUS command, and it now takes the place. And I'm now issuing an alert. Don't use that command. Closing it.

So what I've done here-- this is silly, but it illustrates the concept, which is turn off the default version of a command and write your own version of it. How many times have you ever said, man, I wish the blank command would work like this? We've now got a way to do it.

Let's go back to the auto purge command. So we went out of our way to create this auto purge thing, atp, nobody used it. So what if the blocks were automatically purged out every time somebody saved their drawing? The drawing would be kept clean in an effortless manner. Again, the best standard is the one they don't know they're using. So what I will do is turn off the default qsave command. And that's the little floppy icon that people press a million times a day. And I will build my own key save command.

My key save command will issue a purging instruction. After the purging instruction is completed, I will issue a call to the .qsave command. Is the dot important there? Yes, because if you don't put the dot there, what are you trying to call? You're trying to call something that's undefined. And it'll fail. Princ. Close it out.

Now, when we did this-- I'll never forget it. It was so great. Everybody started calling his desk. Man, is the network slow today? What's going on? Because every time everybody was saving, they were purging. And then it kind of over a process of a few hours, it went away.

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: OK. His question is if you forgot to put the dot there, will it run a recursive self-referencing loop? And the answer is no. Because when you're calling the command line, you're calling only things that are in the command vocabulary. Because this thing here is actually not a command. It's a function. Remember, I said it's as if I've added a command. That's my disclaimer. Yeah, and if you didn't put the dot there, it would just halt.

So this is a very useful way to get in there and customize the command vocabulary. Let's do a couple more things to make this even better. Once we've got this thing debugged, and it works, you can now bulletproof it or run it in stealth mode by turning the command echoing off and back on. The reason that I haven't done a set old, that kind of syntax here is because this is just a binary light switch. It's on or it's off. So what I'm doing here is saying turn or set the command echoing to zero.

Command echoing means when I run a LISP utility, will it shoot the code out the command line? Will it show the user what's actually transpiring? In this case, no, it won't. So they have actually-- they have no idea the purging is occurring. No idea that anything's going on. It just runs.

And then as soon as everything is done running, I'll go ahead and put it back on, which is default state. So I'll put things back the way they ought to be. And you're good. The time to put this in your function, though, is after you know it works. Because if you put this in before you've debugged it, you now cannot see where it screws up. Make sense?

Let's talk a little bit about see CUIs. So did you enjoy the little romp through all the function management and stuff there? That's some pretty cool stuff, isn't it? I use the daylights out of it. Another thing is-- how many of you build your own CUIs and customize stuff in AutoCAD environments? Pretty high correlation. So it's pretty easy for you to go ahead and load in your own CUI file from LISP utilities. And you'll notice that that is using the MENU call, not MENULOAD, because MENU globally loads the whole CUI file as opposed to a partial one. So MENU, the CUI file. Note that I put a significant drive in path there. So yes, you could point to somewhere that's not in a support folder. Note double backslashes.

Once that CUI has been read into memory, what kinds of things are available from inside the CUI file? Workspaces. So you can now instruct a particular workspace to be made current using this syntax. And the only assumption is the workspace better exist, otherwise it will error out. I say this because you may have noticed we had a few laughs it at the user's expense, correct? But there are a lot of what I like to call power or super users out there that are very valid. They help me debug stuff.

I don't want to artificially limit them. In fact, some of them are pretty LISP aware themselves. So I want to give them ways that they can have their own customized environment in a way that won't butt heads with our standardized environment. So what I'd like to do now is just give you a suggested structure for this. You don't have to use this, but it's something that I've been using for years, and it tends to work.

So what I will say to my users is, look, we'll just agree here between you and me there's going to be a way that you can load your own custom LISP code, your own custom CUI file, your own custom workspaces, I don't care. But the agreement is going to be you have to have it named in a certain way, and I'll automatically load it in for you. So as long as you just name your stuff right, I'll let you do whatever it is that you want.

What I'm going to do is I'm going to have them put any of their custom code in a file called USER.LSP. That's the agreement. It needs to be in a support folder, someplace where

AutoCAD can find it and resolve the file. It's their file. I'll not write over it, nor will I check it for accuracy. If it crashes the system, OK. It's their file. It crashed the system. I'm just going to find it and load it if it's there.

Here's how that looks. So you'll notice that I've introduced an if statement there with a findfile function. So if I find the file USER.LSP, I will load it. Notice how that closes down here.

Same trick for the CUI. If you want to have your own peculiar CUI file, that's fine. Just call it user.cuix. If I find it-- doggone it. I hit the wrong button. There we go. If I find it, I will not load it, but I will access the command prompt, call menu, and pull it into memory space. And then they can do whatever they want from there. Your power users would buy this, right? Yeah, I haven't found anybody who will argue with it yet. That's just a nice way-- do what you want, just name it this way, and I promise I won't write over the top of it or blow it away or anything. It's up to them.

You need to tell them, though, if you're running your own CUI file and creating your own workspaces and you're setting up your own desktop, if you totally screw up your CUI file, that's on you. I sure hope you have it backed up, because I'm not tracking it. It's not part of our standard. But seems to be a really good compromise. And incidentally, I actually kind of used this trick a lot of times if I'm developing new code that's not really going to be rolled out for everybody yet. I'll just put it in my own user.lsp, and that way I can load it in and test it that way. And then once I know it works, I can just paste that code over into the global ACADDOC file, and at least I'll know that it works and that it plays nice with all the other functions that I've got in the system. So that's simple, but it's as a good way of any to handle it.

I'm going to introduce here the PROGN function, which is if I find a user.cui file-- maybe they want to load in their user CUI and their own peculiar workspace. Now, the limitation with an if statement is an if statement can only issue one result. So it's like if then else. If, then. So if we have two instructions that we want to execute, we have to group them together with a PROGN or progn grouping. That's kind of archaic. But there it is. It's just the way LISP works.

How could I take this whole happy party up to the network and support everybody at one time? Well, let's think of it this way. If I create the code, and I know that it works, and I tested it on my machine, then I take that code and clone it up to a network drive where everybody reads from it, how many points of control do I now have to change everything? One, and that's about the right number. My goal I will tell you is never to walk around to 50 machines with a memory stick. I don't know how many of you guys have done that. I used to do that with a floppy back in the old days, so I date myself.

Here's how I do it. I will simply set-- on the user's machine-- I will go to their ACADDOC.LSP file on their machine, and I will set up a list path. And that points to a network folder somewhere. Yeah. All right. This now allows me to load files from that lisp path very easily. See how I'm doing that? I introduced a thing called a string concatenation here. Strcat, which puts that path together with the file name itself. So you see how I put a path back-- or a backslash and the end of that? String concatenate, concatenate meaning to put together. So this way, I can very easily point to a network folder. Should that network folder have to change, the worst case is that I have to maybe go back to the user's machine and replace their ACADDOC.LSP maybe about every time I update once a year or something like that. Yes?

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: Yes. So his question is if it was [INAUDIBLE] like server 1 or something, it would be like server 1 back, back. So that will now become back, back, back, back. So it'll be four backslashes instead of the two. But, yeah, that's fine. I will tell you, my preference having done this over the years is to get the map drive letters, because I never know when IT's going to bail out and change servers. I get those panic phone calls that you all the printer paths are busted and such. So I really-- I know it's low tech, but I'm more and more a fan of the map drive names for that reason.

What you can also do, alternately, is you can just put everything on the network folder, and you can configure the user's machine to simply point there. And in this case, there is no local user file. They simply point to the net. That's it. And that's probably the easiest way to do it honestly.

Everybody's now updated on the fly. If you think about it, it's pretty straightforward. The goal is to have everything in one spot. A couple of little utilities and things I wanted to mention here, and I think we're doing OK. We're doing all right on time.

So this is a utility that's called the VLIDE, or the visual LISP integrated development environment. Say that five times fast. So a little window here that pops up with a dollar sign, and you can enter syntax into it like left paraenthesis, function, [INAUDIBLE] will kind of show you if you have imbalanced parentheses. It's got some kind of cool functions in it. Personally, I just kind of use a programmer's editor, but I wanted to show you this just for completeness.

So you can write code in this window if you like, and then save it. Here's the trick that I do use all the time though. If you go into this VLIDE window, and you enter in something like this-- vlisp-compile-- that's a contraction mark, not a quote. St, which means standard mode. And then you put in the name of the LSP file that you've written here, close it, and hit Enter, what will be produced is a file-- MYPROG.FAS, which is what's called fast load format. It's compiled, which means nobody can do what to it? Change it. So this is a way to compile and secure your code.

If you write your code, get it stable, compile it in the VLIDE editor like this, you can then put the FAS files out on the network, and you have got it locked. The worst thing that could happen is somebody could delete the FAS file, but you could put it in a read only folder, and then that can't even happen, can it? You've got it now. What's the crucial, crucial mistake you never want to make though? Never lose the LSP file, because the LSP file is your source code, right? I'm sure there's a way-- there's some Russian hacker somewhere who can reverse and code an FAS file, but I don't know how.

So if you lose the .LSP file, you're in a world of hurt. You are. But this is a great way to compile your code, and then deploy it out onto a network sourcing, and guarantee that nobody's going to fiddle with it. Because a lot of times, it's not only user tampering. A lot of times there's company trade secret stuff in some of these programs you write. You don't really want that falling into enemy hands. So it's for intellectual property protection as well. Yes?

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: Yes. Well, actually whichever-- his question is does the FAS take precedence over any LSP that you would load? And the answer would be whichever loads last wins. Yes. What I would do let's say if I were using a network load in structure, it would not say init.lsp. It would say init.fas. Good question. Should revise my slide deck for that. That was a very good question.

OK. Let's move here. So I've kind of give-- and I realize we've gone all over the place, but I hope some of these things have been kind of cool and have made you think about some ways that you could now introduce some of these functions and tools into your environment. The last thing that I wanted to go over is the vexingly difficult problem of dealing with AutoCAD profiles. How many of you have difficulties with users screwing up their profiles, changing paths, stuff like that?

So what I would like to do is have a way to manage AutoCAD's configuration without having to use profiles. And it turns out LISP gives you a great way to do it. There's a few steps involved, which I will run through in sequence for you. Not that hard, you just got to know what to do in what order.

If you could set printer directories, palette locations, stuff like that without having to create profiles, not only would it be more consistent, but you'd never have to go to the user's machine. And that to me is the real magic. I don't want to go lay hands on that machine. Do any of you guys have the problem when you go somewhere to do a 30 second operation on a user machine, and you get marooned there, right? They won't let you leave. You have this? Happens to me.

So what I'm going to do is I'm going to get a new set of functions. We had GETVAR and SETVAR before, right? Well, now we're going to have GETENV and SETENV. And rather than reading or writing to AutoCAD's local variable section, we're going to be reading in writing from what? The registry. And a sense of awe overcame the room.

So anytime I say registry, you should be fearful potentially. Yeah, I don't know what you're doing here. Not that hard, but what we-- the key is AutoCAD will protect us. And here's why. What you don't really understand, because AutoCAD never tells you this, but when the Autodesk application fires up it, it reserves a narrow, skinny, little slice of the registry that it can read and write into without any administrative level access. Think about the power of that. But it's only going into what the current user profile is.

So if they're currently in vanilla-- if they're currently in my profile 2, they can only access the current profile that they're in. So it's not like you can do huge harm to the registry. So there's no need for overwhelming paranoia, but a little paranoia is always good.

If you were to go into the registry using REGEDIT-- how many of you can do this, by the way? All right. If you're assuming your IT department doesn't lock you down. What you'll do is kind of trudge your way down through the My Software tree. And you'll eventually find your way into where the Autodesk stuff is. And then you'll eventually find your way into the current version of AutoCAD. And then you'll eventually find the tab that says Profiles. And you'll eventually find the one that's current, which when I created this screen grab, was the vanilla profile.

You then expand that vanilla profile tab, and it shows you everything that AutoCAD knows about that profile. The one that is of interest most times is the tab that's called General. So if you expand this, you'll now see pretty much all the stuff that's in the Files tab when you're manipulating a profile. Believe it or not, this is how you set a registry key in AutoLISP. It's not that hard. Let me go backwards.

So here, we've opened the General tab. And this is what we see. Over here, one of them is the printer config directory. And you see over here it says user profile. And then it's got a path associated behind it. What I need to know is exactly what that registry key is, case sensitive. And as soon as I know exactly what that's called and exactly how it's spelled, I can then write to it using the set environment function like this.

So right after I run this LISP statement, what's the printer path set to? Z acad Plotters. What happened to the way it used to be set? Gone. So that may be not the best way to handle it, right? Because we may be losing plot configuration folders that we really wanted. This down here is better.

We'll set the PrinterConfigDir, and we'll do it like this. So again, I'm using a string concatenation or strcat. And I am putting Z acad Plotters, semi-colon-- why semi-colon? Because that's the way you put multiple paths together. I don't know if you ever noticed that, but in the Files tab, if there's multiple paths, that's how they put them together. And then I will get the current printer configuration, so get environment, PrinterConfigDir. Two levels apparence, so which one analyzes first? The get, which will then get joined to here, which will then get set using this.

So there's actually three levels of parentheses happening here. Getting, setting equal to, then setting the environment. Solving your way inside out. So this will actually make sure that the Z ACAD Plotters folder will be first in your print folders configuration. So that's where AutoCAD will look first. If it doesn't find the printer configuration there, it'll fall back to any other folders that may be defined.

What's the problem with this? It's not obvious.

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: Can't-- yeah. He's going in the right way. I can't reset it back to the way it was, because it's been pushed to the registry. It's a good point. The other thing is if I do this every time AutoCAD starts a new drawing session, what's going to happen every time it starts? It's going to put Z ACAD Plotters on the front of it every single time. So the 43rd time I open AutoCAD, what does that path look like? It's a mile long, right? Z ACAD Plotters is in there 43 times, which I don't want.

What I want to do is have a way to check it. Has the printer configuration directory been set to include Z ACAD Plotters? If it has not, I want to go ahead and do it. But if it has already been set, I want to skip it. Does that make sense? Here's what that code looks like. There it is. And yeah, so this cutting the sides of the screen off is not ideal here. So what I'm saying is if not, I get the environment here. So I get the current printer configuration directory. Then I do a wildcard match. See this set of parentheses here? Wildcard match. So if the current printer configuration directory has Z ACAD Plotters in it.

If it does not have that in it-- see how I did that? I logically negated it. If it does not pass this test, then I will go ahead and put that on the front of it and push it into the registry. This is a recipe you can use to set any kind of path in AutoCAD's registry and test it to make sure that it only gets initialized once. And I'll put a cheat sheet of some commonly used variables up here for you in a second. Yes?

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: It goes to the local user.

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: It will always run, because if AutoCAD doesn't have permission to write into that segment of the registry, AutoCAD will fault. Have you ever run AutoCAD, and it won't hold a configuration? That's the problem. Yeah. So if AutoCAD is running correctly anyhow, this will work.

Here's that cheat sheet I was talking about. So the support path is-- here's PrinterConfigDir for context. That's the one we've been talking about. The general support path is ACAD. The drivers path, ACADDRV. Here's where the tool palettes come from. Template files, plot styles, current CUI, the current workspace. Here's the trusted paths. Now, this is by all means not a complete listing of variables, but I did show you how to trudge in there in the registry editor and look them up, right? So you could do a little bit of sloothing. But these are the ones-- by and large at least 99% of what I do is just making sure that the printers and the support path files are correct. If you never use anything other than that, you're probably good to go.

That's kind of theoretical. Did that make sense? Let's draw some conclusions, and maybe we might have a couple of things here. And then if you guys want to have any questions or something, feel free to come up afterwards. Hopefully, no matter what your skill level was in AutoLISP, maybe you learned a couple of things. Was that true? OK. I think this is a fantastic management tool to control, not only because of what you can do to the user command vocabulary to achieve standardization, but just being able to get around the profiles and directly get into the registry stuff is profoundly powerful.

I get the question, I'm a CAD manager, should I learn LISP? Yes. Absolutely. Everybody who is a CAD manager should know something about this language. Yeah.

AUDIENCE: I've got a question for everybody. How many of you in here still have users that type in [INAUDIBLE]?

ROBERT GREEN: Typers?

AUDIENCE: [INAUDIBLE]

ROBERT GREEN: Yeah. They don't use the shortcut?

AUDIENCE: No.

ROBERT GREEN: I got to tell you, man, I still type though, because I don't care what they do, I'll never get in the erase command faster than E-spacebar. Never. There's no user interface enhancement they can do that's quicker than that.

Kind of closing out. I'll do a couple of things, and these are in the handouts. First of all, did everybody figure out how to get into their AU account and get handouts and stuff? OK. So you can go there. I will go ahead, and I had one correction to make with a double backslash. I saw that.

The other thing that I will do is I will have my code samples typed out for you in files so that you don't have to type this stuff out yourself. So you can go in and pull all that stuff down. I also put it in the handout I've got it available on my website. So if you want to get there as opposed to going through the AU portal, you can. Early next week-- probably Monday-- I'll go ahead and send out a class email just reminding you that the files are there. So if you get an email from me, it's not spam. It's legit.

I'll go ahead and put my email up here. I'm happy to hear from you. The only thing I ask is please don't attach a two megabyte LISP file and say, what's wrong with this? Because that can take quite a while. Did everybody learn something of use here? Did you enjoy it? Good deal, man. Well, thanks for coming, and I enjoyed having you.