0:0:0.0 --> 0:0:1.280  
Conf Room STUDIO C/3038 (14)  
It it knows you're like it's.

0:0:3.920 --> 0:0:8.610  
Dylan Bevan  
Right on, let me share my screen. Thank you all for coming. F# is one of my passion.

0:0:9.600 --> 0:0:18.450  
Dylan Bevan  
Passion things. I think I do and evangelization, evangelization session for pretty much everywhere I go to and I have had zero traction so far, but I live in hope.

0:0:19.190 --> 0:0:20.360  
Dylan Bevan  
Eternally optimistic.

0:0:22.570 --> 0:0:28.660  
Dylan Bevan  
Cool. So I think first thing I'd prefix is if you haven't looked at a functional language, you 100% should.

0:0:30.330 --> 0:0:35.160  
Dylan Bevan  
You can actually do a lot of functional stuff in C# because they keep stealing more and more stuff from us. Sharp.

0:0:35.430 --> 0:0:55.900  
Dylan Bevan  
I'm I just find in general if you think about things in functions, your programs last a lot longer than if you think about them in terms of object. That's purely because as we all know, our definition of objects in object oriented programming is poorly taught in schools, and then stuff inevitably changes over time, which leads to more friction in object oriented.

0:0:57.300 --> 0:1:7.300  
Dylan Bevan  
So with that out the way first thing is if you just do a new F# program, it should say a sharps are fully .net compliant language, so there's nothing that's C# can do that F can't.

0:1:9.890 --> 0:1:15.210  
Dylan Bevan  
It's just sometimes if shop like it's not very good at dynamic program because it's incredibly strongly typed.

0:1:15.980 --> 0:1:17.310  
Dylan Bevan  
Despite the way it looks.

0:1:18.370 --> 0:1:27.200  
Dylan Bevan  
So we'll get to annoy SEC, but the first neat thing is from within Visual Studio. If you take a snippet of code and you just highlight it, you can press alt and enter.

0:1:27.990 --> 0:1:32.510  
Dylan Bevan  
And that will send it to something called F# Interactive, where it will just go run it so you can see this is the result.

0:1:34.220 --> 0:1:48.820  
Dylan Bevan  
That's called the variable it. Everything in in UM, F# has to be bound to a value. Unit is equivalent to void, right? So print function returns nothing, so it's a void function, effectively, which we call unit, and this is the result.

0:1:49.540 --> 0:1:50.470  
Dylan Bevan  
So we can get rid of that.

0:1:52.110 --> 0:1:59.900  
Dylan Bevan  
And the first thing I'll do is just copy this part across. So this is the strongly typed Ness my God, if I can copy and paste, there's like.

0:2:0.890 --> 0:2:3.510  
Dylan Bevan  
Beginning demo fail like that. There we go.

0:2:4.480 --> 0:2:8.430  
Dylan Bevan  
So we have this function, so let is kind of like.

0:2:9.470 --> 0:2:10.620  
Dylan Bevan  
The I guess.

0:2:11.520 --> 0:2:13.180  
Dylan Bevan  
So it's saying let print number.

0:2:13.920 --> 0:2:23.280  
Dylan Bevan  
Which takes one argument of X equals this. So in C# equivalent, this is the the same as public. Actually this is void.

0:2:24.260 --> 0:2:25.30  
Dylan Bevan  
Number.

0:2:26.920 --> 0:2:29.340  
Dylan Bevan  
Number in text.

0:2:31.270 --> 0:2:35.440  
Dylan Bevan  
And you can do the fancy arrow function if you wanted to, and then basically a fancy sole.

0:2:36.530 --> 0:2:37.510  
Dylan Bevan  
Not like line.

0:2:39.440 --> 0:2:41.620  
Dylan Bevan  
Right, right. Well, number is X.

0:2:42.320 --> 0:2:45.760  
Dylan Bevan  
So it's a lot more succinct writing in in F#.

0:2:46.450 --> 0:2:47.890  
Dylan Bevan  
And for hover over X.

0:2:48.570 --> 0:3:9.190  
Dylan Bevan  
I don't know if you can really see that, but visual studios popping up it's saying the value of X and then colon is int, so the type of X is integer. Now is able to infer that because I've put percentage I here which stands for integer. So this is your string format. So if I were to try and pass in another variable like a string, I'd get a compilation error even though.

0:3:10.820 --> 0:3:20.870  
Dylan Bevan  
In C#, for example, you you typically just say this is parentheses X right in your string form. You do that lovely dollar and then you do like blah blah X.

0:3:21.940 --> 0:3:27.320  
Dylan Bevan  
And it doesn't matter if the if the type is incorrect in C#, it won't care. Whereas if sharps much more strongly typed.

0:3:28.60 --> 0:3:35.450  
Dylan Bevan  
Clear down. I can alt and enter this and now print number is stored in here so I can recall print number with 100. Help.

0:3:36.290 --> 0:3:40.990  
Dylan Bevan  
I've got semicolons and interactive and there we go. Number is 100, so it's printed out that function.

0:3:42.70 --> 0:3:57.240  
Dylan Bevan  
So typically you don't see types in your code in off, and that's because the guy who came up with generics for C# two is called Don Simon. He's like the guy who basically started the implementation of F# and still leads, I believe, to this day.

0:3:58.800 --> 0:4:0.890  
Dylan Bevan  
So it's very heavily genericized.

0:4:2.930 --> 0:4:7.880  
Dylan Bevan  
Interesting thing about all the functions in F is they all only ever take one argument.

0:4:9.340 --> 0:4:9.820  
Dylan Bevan  
So.

0:4:10.870 --> 0:4:23.940  
Dylan Bevan  
You can do this right, add X&Y and this is gonna return the result of X&Y file for over these X&Y and int because plus even though there's multiple overloads for it, it'll default down to X. Sorry just.

0:4:24.900 --> 0:4:28.990  
Dylan Bevan  
Umm, So what this does under the hood? Is it basically curries the function.

0:4:30.20 --> 0:4:33.210  
Dylan Bevan  
So you'll hear currying a lot when you talk about functional languages.

0:4:34.10 --> 0:4:39.530  
Dylan Bevan  
And the way that the compiler will actually write this function for you under the covers is like this.

0:4:40.440 --> 0:4:49.70  
Dylan Bevan  
Right. So you have a function that takes a parameter X and that returns a function which takes a parameter Y and then it'll sum X&Y together.

0:4:50.590 --> 0:4:52.680  
Dylan Bevan  
So if I do, I'll then enter here.

0:4:53.670 --> 0:5:1.360  
Dylan Bevan  
Obviously I can do add 10 and five or I keep forgetting those. That's gonna give me 15 the curried function.

0:5:2.410 --> 0:5:2.730  
Dylan Bevan  
The.

0:5:3.550 --> 0:5:11.560  
Dylan Bevan  
Even though the signature looks different int, so it takes an int and it returns a function that takes an int and then that will return an int. So I can say add Harry.

0:5:12.490 --> 0:5:16.280  
Dylan Bevan  
Uh sounding 5 forget my semi colon so you get exactly the same result.

0:5:18.20 --> 0:5:27.410  
Dylan Bevan  
So you might think that's completely like whack a doodle just having a single argument to every function, but it actually means that you can do a heck of a lot of reasoning about functions.

0:5:29.560 --> 0:5:36.470  
Dylan Bevan  
Within the language, so they say it in F#. If it compiles, it will work because you can be a lot more.

0:5:37.230 --> 0:5:39.860  
Dylan Bevan  
You can make a lot more guarantees about how the code is going to work.

0:5:41.140 --> 0:5:44.980  
Dylan Bevan  
But mostly it boils down to the fact that each function only takes one argument.

0:5:48.10 --> 0:5:48.460  
Dylan Bevan  
OK.

0:5:49.150 --> 0:5:52.120  
Dylan Bevan  
So there's another terminology, so this is carrying.

0:5:53.500 --> 0:5:58.490  
Dylan Bevan  
There's another terminology which is called partial applications, so this is a partially applied function.

0:5:59.690 --> 0:6:5.210  
Dylan Bevan  
So this is our function add up here and it takes 2 arguments, X&Y effectively.

0:6:6.180 --> 0:6:7.330  
Dylan Bevan  
And I'm only giving it one.

0:6:8.440 --> 0:6:23.100  
Dylan Bevan  
And if we have a look at the signature of ad partial down here saying it'll take an integer and return an integer, the whole, I will say F# looks really really weird until you get used to the syntax. So if this looks like completely alien stuff, that's totally normal.

0:6:24.890 --> 0:6:26.620  
Dylan Bevan  
But now I can call add partial.

0:6:28.740 --> 0:6:34.750  
Dylan Bevan  
And just give it 5 and that will give me the 15 because I've basically had a closure effectively around 10.

0:6:37.710 --> 0:6:40.830  
Dylan Bevan  
Umm, so this comes in.

0:6:41.680 --> 0:6:42.990  
Dylan Bevan  
Super useful.

0:6:43.950 --> 0:7:4.880  
Dylan Bevan  
If you think about, let me just see if I've got an example of this a wee bit later. I kind of do, but I'll I'll cut ahead to this as to where this is good because I've put in the the like pro forma for this meeting like everything in F# is functions. All those fancy patterns are used to get in C# disappear. So this is how you do dependency injection effectively.

0:7:5.750 --> 0:7:10.390  
Dylan Bevan  
So if you think you've got some, let's say like business layer.

0:7:13.480 --> 0:7:14.160  
Dylan Bevan  
Function.

0:7:14.880 --> 0:7:18.590  
Dylan Bevan  
Which takes their repository repository.

0:7:19.650 --> 0:7:23.660  
Dylan Bevan  
And an item and it's gonna do some validation and save it right?

0:7:25.750 --> 0:7:27.850  
Dylan Bevan  
You can actually say in here that repository.

0:7:29.290 --> 0:7:35.350  
Dylan Bevan  
This is gonna take a generic which case we'd say prime A and it's gonna return a blue all for example.

0:7:37.710 --> 0:7:44.280  
Dylan Bevan  
So this is effectively an interface declaration and then item. Here we're gonna say is is.

0:7:45.440 --> 0:7:47.170  
Dylan Bevan  
And generative of type A.

0:7:48.130 --> 0:7:57.70  
Dylan Bevan  
So in in F# you use a BCD instead of TT1T2. All that kind of stuff. So now I can just say repository with item.

0:7:57.950 --> 0:7:59.300  
Dylan Bevan  
Right. And that's all lovely function.

0:8:0.380 --> 0:8:15.250  
Dylan Bevan  
If you want to test your business layer function now, you just pass in a function that's gonna return true or false depending on what you wanna do here and when it comes to writing the actual implementation, then you just wrap. However, your code works in this.

0:8:16.20 --> 0:8:18.660  
Dylan Bevan  
And then if you wanted to do like a let. Oops.

0:8:20.340 --> 0:8:22.10  
Dylan Bevan  
Let's test function.

0:8:22.970 --> 0:8:29.120  
Dylan Bevan  
Equals business layer function and then I'm just gonna say function like given X always gonna return true.

0:8:32.160 --> 0:8:33.980  
Dylan Bevan  
And talking about X.

0:8:35.580 --> 0:8:39.20  
Dylan Bevan  
Ohh, shouldn't have to do that. I wonder why it's doing that. Yeah, it's not happy.

0:8:40.350 --> 0:8:52.640  
Dylan Bevan  
In and it ohh generic type blah blah blah. OK so yeah, we'd have to give it the generic type. Yeah. So we'll just ignore that for a week. SEC. But essentially this is your test function and then obviously if you wanted to do your real function.

0:8:53.810 --> 0:9:0.660  
Dylan Bevan  
And you'd be business layer function and then you pass it your like EB functional wherever you are sending it to.

0:9:3.60 --> 0:9:12.870  
Dylan Bevan  
So it makes it very easy. Like I say, you can reason about it. So now you're unit test don't have to have mocks and all these things because you're literally just passing in something that's gonna do explicitly what you wanted to do.

0:9:16.560 --> 0:9:21.490  
Dylan Bevan  
Alright, next cool thing is all data structures are immutable.

0:9:22.940 --> 0:9:32.530  
Dylan Bevan  
So this is how you declare what would be a class in C#, and these are properties, but this is totally immutable. So if I create a sample address.

0:9:33.810 --> 0:9:35.540  
Dylan Bevan  
And I'm just gonna pop this down here.

0:9:36.430 --> 0:9:38.0  
Dylan Bevan  
And I've got blah blah, blah blah blah.

0:9:39.440 --> 0:9:45.650  
Dylan Bevan  
You can see this type inference in action right? So like I said, I've shop is very strongly typed, so nowhere in here by said this is an address.

0:9:46.370 --> 0:9:58.820  
Dylan Bevan  
But Visual Studio knows that it's an address because it looks at all the code that's written and it says OK, you've passed me in a house number and the only thing that has a house number in it is this class. Therefore, you must be using this class.

0:9:59.940 --> 0:10:1.510  
Dylan Bevan  
And in case you're.

0:10:2.860 --> 0:10:3.880  
Dylan Bevan  
Concerned, right?

0:10:5.430 --> 0:10:14.580  
Dylan Bevan  
It's now a red squiggly and it says there's no assignment given to street. You have to give an assignment here, and if I do my control space, you can see that I have all the properties available to me.

0:10:15.870 --> 0:10:16.800  
Dylan Bevan  
I'm from.

0:10:19.260 --> 0:10:23.30  
Dylan Bevan  
The entire sentence, so it it's 100% now is that this is what this is.

0:10:24.600 --> 0:10:33.390  
Dylan Bevan  
So let's say I wanted to change this. So I wanna do sample address. I wanna change the House number to #11 so I can do House number can cause 11.

0:10:35.100 --> 0:10:42.170  
Dylan Bevan  
So there's no compiler error here, but it's got a green squiggly and it's saying the result of this equality expression has type bool.

0:10:43.340 --> 0:10:48.230  
Dylan Bevan  
So F# is a foundation has its foundation in mathematics.

0:10:49.270 --> 0:10:50.880  
Dylan Bevan  
So equals in.

0:10:51.680 --> 0:10:56.280  
Dylan Bevan  
An evaluation expression means equals the same way as it does in maths. Don't have to do double equals.

0:10:57.230 --> 0:11:8.720  
Dylan Bevan  
So the way that you would assign a value in F# is to give it a pointy arrow. That's kind of like saying push 11 into this valley. Now you can see it's complaining cause it says this field is not mutable.

0:11:10.260 --> 0:11:13.730  
Dylan Bevan  
Now you could do it by adding mutable keyword.

0:11:14.490 --> 0:11:18.280  
Dylan Bevan  
And you can see it glows up bright yellow because this is an awful thing to do.

0:11:19.560 --> 0:11:21.670  
Dylan Bevan  
Right. You never wanted to do this kind of stuff.

0:11:22.620 --> 0:11:31.750  
Dylan Bevan  
So how this is crazy, right? We just wanna change the House number to 11. So how do we do that? So I'm gonna do like new house equals. And now I'm gonna do sample adrenals.

0:11:32.810 --> 0:11:33.750  
Dylan Bevan  
Nonpolar dress.

0:11:35.540 --> 0:11:38.90  
Dylan Bevan  
With and then I'm going to say house number.

0:11:40.230 --> 0:11:41.380  
Dylan Bevan  
Equals 11.

0:11:43.420 --> 0:11:50.530  
Dylan Bevan  
And that's how you do it now. It's basically gonna copy everything across. So every type in F# comes with.

0:11:51.750 --> 0:11:53.40  
Dylan Bevan  
A copy constructor.

0:11:54.440 --> 0:12:0.630  
Dylan Bevan  
Now let's just do this. I'm gonna call this sample address prime, because that's typically how we'd write things in.

0:12:2.340 --> 0:12:5.30  
Dylan Bevan  
In F# as well, if we're like duplicating some things.

0:12:5.720 --> 0:12:19.380  
Dylan Bevan  
So now I'll say sample address equal oops equals sample address fine and then I mean not get to that just yet. Let's do print function and now we're gonna give it a bowl and then this is the function that we actually want to give it.

0:12:21.30 --> 0:12:26.30  
Dylan Bevan  
So let me just pop pop these ohin gonna put the address in class. Let's get that going.

0:12:27.170 --> 0:12:28.720  
Dylan Bevan  
Then we'll put these in.

0:12:29.800 --> 0:12:48.280  
Dylan Bevan  
And then I'll say are these equal and the answer is true. So by default F# does equality comparisons, not referential structural equality comparisons, not referential equality, which is super amazing. I don't know why C# still has referential equality.

0:12:49.740 --> 0:12:55.270  
Dylan Bevan  
And C# seven I think is actually having these immutable data structures in there called records in C#.

0:12:56.680 --> 0:13:1.630  
Dylan Bevan  
So these things are coming into C#. You should totally use them. It's 100% way better.

0:13:3.560 --> 0:13:13.910  
Dylan Bevan  
So, since you're all in America, this is a particularly interesting feature that they have in their language called units of measure.

0:13:16.590 --> 0:13:17.80  
Dylan Bevan  
So.

0:13:16.770 --> 0:13:17.700  
Conf Room STUDIO C/3038 (14)  
Freedom units.

0:13:18.180 --> 0:13:19.270  
Dylan Bevan  
You know what? This you're gonna go.

0:13:20.800 --> 0:13:31.600  
Dylan Bevan  
So I can just use this attribute and attributes. You can do it saying this C#, but most people just put it like in line like this. So here I have a measure and the type is meters and here I have a measure of type seconds.

0:13:32.660 --> 0:13:34.770  
Dylan Bevan  
So now I'm going to declare a distance.

0:13:36.260 --> 0:13:37.590  
Dylan Bevan  
Of 100 meters.

0:13:40.340 --> 0:13:47.620  
Dylan Bevan  
We need one type of type of unit of measure for spelling as well, so I'm not sure if it's ERS in America or if it's Rs.

0:13:47.480 --> 0:13:49.750  
Conf Room STUDIO C/3038 (14)  
Yeah, it's it's it's ERS.

0:13:49.890 --> 0:13:53.710  
Dylan Bevan  
Yeah. So yeah, so that's another typo. You can see my lovely spell checker is not complaining.

0:13:55.210 --> 0:13:58.160  
Dylan Bevan  
And then I have time in 9.58 seconds.

0:13:59.550 --> 0:14:1.670  
Dylan Bevan  
So this means that I can't.

0:14:2.380 --> 0:14:14.910  
Dylan Bevan  
If if you have a, what was it? One of the space shuttles blew up because the Russians had used or the rest of the world had used millimeters as the measurement. And I think the US had used inches.

0:14:15.670 --> 0:14:37.260  
Dylan Bevan  
And no one had caught the caught the mistake, right? Cause everyone who's reviewing it all looks great. No one thought that anyone else is using different measures. And in this case you can write your code to be type safe. If you wanna do it. The important thing is this is as lovely as Java generics. They get completely erased when your project gets compiled. So it's only a compile time check. You can't then use units of measure at runtime.

0:14:39.310 --> 0:14:46.370  
Dylan Bevan  
Another beautiful thing with F# is that if you write double ticks, you can write.

0:14:47.110 --> 0:14:48.640  
Dylan Bevan  
And what I've done so I can say that.

0:14:50.210 --> 0:14:51.410  
Dylan Bevan  
My magic.

0:14:52.960 --> 0:14:53.700  
Dylan Bevan  
Functions.

0:14:54.360 --> 0:14:59.510  
Dylan Bevan  
X = X + X and It's totally happy. Now I can call my magic function.

0:15:1.70 --> 0:15:1.590  
Dylan Bevan  
Like this?

0:15:3.930 --> 0:15:5.390  
Dylan Bevan  
So why does that matter?

0:15:4.800 --> 0:15:6.820  
Conf Room STUDIO C/3038 (14)  
Does that mean I can have emojis there?

0:15:7.140 --> 0:15:17.170  
Dylan Bevan  
Yes, I actually did. So when I gave this talk at the coalition, somebody asked exactly the same thing. So I wrote a serializer which was flipping a table and then deserializer was putting the table back.

0:15:19.940 --> 0:15:36.490  
Dylan Bevan  
So let's figure out you saying bolts speed because this is absolutely bonkers. Everyone knows who's the fastest person on the planet, but when you actually see how fast he's going crazy. So here we have our distance, which is in meters and time. So the question is, what is the type of you saying speed.

0:15:41.520 --> 0:15:41.870  
Conf Room STUDIO C/3038 (14)  
2nd.

0:15:38.940 --> 0:15:42.610  
Dylan Bevan  
So if we hover over, it's gonna tell us it's a float in meters per second.

0:15:43.550 --> 0:15:47.740  
Dylan Bevan  
Right. So it'll actually do these calculations for you on the fly and then it won't let you.

0:15:48.440 --> 0:16:0.220  
Dylan Bevan  
Umm, you know. However you chain these things together, you always have to have your units in order to make it work. So if you're doing something that's scientific or mathematical, these things come in superly badly. To make sure that nothing.

0:16:0.700 --> 0:16:3.500  
Dylan Bevan  
Umm is screwed up.

0:16:4.260 --> 0:16:5.550  
Conf Room STUDIO C/3038 (14)  
Dylan, I have a question.

0:16:5.720 --> 0:16:6.170  
Dylan Bevan  
Yes.

0:16:6.620 --> 0:16:16.70  
Conf Room STUDIO C/3038 (14)  
The The You you mentioned that like it automatically picks up like based on your type, it automatically picks up when you define an instance of of that type.

0:16:17.370 --> 0:16:17.690  
Conf Room STUDIO C/3038 (14)  
But.

0:16:24.580 --> 0:16:25.320  
Dylan Bevan  
Yes it.

0:16:18.390 --> 0:16:25.900  
Conf Room STUDIO C/3038 (14)  
What if you so will it not let you define another type with the same field names in it, so that it could be confused.

0:16:26.350 --> 0:16:29.50  
Dylan Bevan  
Yeah. So you could do type small address.

0:16:30.340 --> 0:16:32.30  
Dylan Bevan  
And then you can do.

0:16:33.870 --> 0:16:37.620  
Dylan Bevan  
I came here. Let's just copy this first things for example.

0:16:40.790 --> 0:16:45.880  
Dylan Bevan  
It's still happy because you have to declare every single property in here. None of them are optional.

0:16:47.620 --> 0:16:58.650  
Dylan Bevan  
So if you Type House number 31 St two, you could now close this off and it would be happy because well, now it's saying it's unhappy because we're trying to compare two different types, smaller address and address.

0:16:59.710 --> 0:17:0.10  
Dylan Bevan  
Umm.

0:16:58.760 --> 0:17:3.260  
Conf Room STUDIO C/3038 (14)  
Yeah. OK. So it'll it'll go to the one that it sees that it thinks matches.

0:17:3.720 --> 0:17:11.850  
Dylan Bevan  
Absolutely. And then as soon as you put city in. So if I was to close off here, then we can figure out the type and then it's gonna complain. Here you're missing all the other properties.

0:17:16.170 --> 0:17:16.550  
Dylan Bevan  
The.

0:17:15.770 --> 0:17:19.500  
Conf Room STUDIO C/3038 (14)  
If you have two identical type, what if you have two identical types? I mean, yeah.

0:17:20.110 --> 0:17:25.80  
Dylan Bevan  
Yeah. So if you were to do type address too equals one.

0:17:26.190 --> 0:17:30.370  
Dylan Bevan  
I mean, there would be namespaced differently, right? So you're gonna have that thing there.

0:17:31.550 --> 0:17:34.470  
Dylan Bevan  
Now, it's probably just gonna find the first one actually takes the last one.

0:17:35.130 --> 0:17:43.740  
Dylan Bevan  
But you can specify that this is an address one or an address by doing that. So this colon is the type which is same as if you've done any TypeScript, it's the same sort of thing.

0:17:45.700 --> 0:17:50.370  
Dylan Bevan  
Never had that occur to be honest, but I can see that possibility for it.

0:17:51.980 --> 0:18:2.970  
Dylan Bevan  
OK, so let's get some of these printed in. So let's go stick those things in, and then we're gonna get the same speed in. So he does 10 meters a second.

0:18:4.780 --> 0:18:8.260  
Dylan Bevan  
Which is bonkers. I think. I'm like under 2 meters tall.

0:18:9.200 --> 0:18:11.970  
Dylan Bevan  
So he can do like 5 of me in a second.

0:18:12.830 --> 0:18:13.880  
Dylan Bevan  
Which is just.

0:18:14.530 --> 0:18:15.770  
Dylan Bevan  
I don't know. I think that's just crazy.

0:18:16.500 --> 0:18:25.20  
Dylan Bevan  
OK, so let's print this out. So another thing with F# that's nice is the readability of the language, but there's it's a double edged sword, but we'll we'll get to that. Maybe in a bit.

0:18:25.780 --> 0:18:39.390  
Dylan Bevan  
And so we wanna figure out what you're saying. Speed is. So we're gonna print F here. Now I can do a percentage A which I don't know what the A stands for, but it's basically gonna let F# figure out the best way to display the type because this is still meters per setting. So I've been run that.

0:18:40.300 --> 0:18:43.330  
Dylan Bevan  
And then it just says it's 10.43 blah blah blah blah blah.

0:18:44.600 --> 0:18:58.490  
Dylan Bevan  
So with C#, everything's inside out if you ask me. See, you've got like your function ABCD and then finally you have like your value in there and you see this a lot, right?

0:18:59.600 --> 0:19:10.130  
Dylan Bevan  
So with F# you have a pipe forward operator. So instead of doing that we can type and same speed and then we can just use this which means pipe that into the next function.

0:19:11.40 --> 0:19:14.910  
Dylan Bevan  
And then it's essentially I can just copy this part down, yeah, and say that.

0:19:16.260 --> 0:19:18.240  
Dylan Bevan  
And and run that and it's exactly the same thing.

0:19:19.530 --> 0:19:40.20  
Dylan Bevan  
So we'll get to this in a in a wee bit, but but you can just chain these things together forever and ever and ever and it becomes readable, especially if you're doing something like Lync, like where you'll say filter fold, which is like an aggregate and then do something else with it, save it or whatever you wanna do. And so it reads like English. Whereas if it was in C you'd have like save.

0:19:42.380 --> 0:19:43.690  
Dylan Bevan  
Aggregate filter.

0:19:44.930 --> 0:19:45.640  
Dylan Bevan  
On X.

0:19:50.600 --> 0:20:7.930  
Dylan Bevan  
And the neat thing is so that pipe forward the that I've just erased and you can declare your own operators. So one I did before which drove one of my colleagues mad was this beautiful symbol. So we did a lot of string comparisons.

0:20:9.0 --> 0:20:12.170  
Dylan Bevan  
And so I can do string dot.

0:20:15.700 --> 0:20:16.450  
Dylan Bevan  
I'm happy about that.

0:20:19.180 --> 0:20:21.480  
Dylan Bevan  
I'm gonna be system that's running.

0:20:22.610 --> 0:20:23.210  
Dylan Bevan  
Stopped.

0:20:23.940 --> 0:20:30.330  
Dylan Bevan  
Compare ordinal AD and then whatever the ordinal types are.

0:20:31.840 --> 0:20:37.450  
Dylan Bevan  
That's not gonna give me the string comparison. Potentially string compare really string ohse I'm doing ordinal there.

0:20:38.240 --> 0:20:42.870  
Dylan Bevan  
OK, so now I have this weird looking thing, but now I can say a.

0:20:44.300 --> 0:20:45.90  
Dylan Bevan  
Right a.

0:20:51.840 --> 0:20:53.110  
Dylan Bevan  
It speaks of building metal.

0:20:54.560 --> 0:20:55.820  
Dylan Bevan  
It's all that. Send that in.

0:20:58.770 --> 0:21:0.690  
Dylan Bevan  
Ohh it's 32.

0:21:1.790 --> 0:21:17.420  
Dylan Bevan  
Because I don't know my functions properly, but that anyone know what that function is the proper one to do? Like a string comparison my here we go. Luckily my reference material has it it's string dot equal it's not compared all there we go. So let's do that and we'll just redefine that function.

0:21:18.190 --> 0:21:26.860  
Dylan Bevan  
Pretend I did it right the first time, so now you can see it's true, right? So A is a. When we do that string comparison ordinal case. So this saves you having to write this crap all the time.

0:21:28.540 --> 0:21:45.160  
Dylan Bevan  
So I said earlier the readability of our sharp is really high, but this is the sort of stuff that makes it sort of illegible over time. Or has that learning curve when somebody comes into a new code base if people start abusing this operator definition stuff?

0:21:47.820 --> 0:21:48.730  
Dylan Bevan  
All good so far.

0:21:51.500 --> 0:21:57.830  
Dylan Bevan  
Cool. Uh, we are going to let me see where. Ohh, there's no such thing as null.

0:21:59.770 --> 0:22:1.650  
Dylan Bevan  
Right. So if I were to do.

0:22:3.260 --> 0:22:13.630  
Dylan Bevan  
Yes. So we have our address. I'm going to say that null address, I'm giving it the type of the dress. It knows what that is and I'm saying it's null and it's complaining says does not have null as a proper value.

0:22:15.200 --> 0:22:16.190  
Dylan Bevan  
So this is awesome.

0:22:17.900 --> 0:22:27.330  
Dylan Bevan  
What do you do when you need to know when you give it what's called an option type? Now I think this is in C# since 5 actually that they call it something else in C#.

0:22:28.660 --> 0:22:33.810  
Dylan Bevan  
But here we have an adopt an option of address, and this can be either none or some.

0:22:35.180 --> 0:22:39.540  
Dylan Bevan  
Right. So then if we were to do a function which is to print an address.

0:22:41.660 --> 0:22:42.920  
Dylan Bevan  
Put that down here.

0:22:45.840 --> 0:22:50.170  
Dylan Bevan  
And it takes an option of address because maybe it's the user hasn't filled in their address yet.

0:22:52.310 --> 0:22:59.120  
Dylan Bevan  
If I do this print F, the address dot St and use the string, this won't work.

0:22:59.990 --> 0:23:5.900  
Dylan Bevan  
Right, because option doesn't define the field St cause option is like a wrapping type.

0:23:6.660 --> 0:23:10.610  
Dylan Bevan  
Effectively So what you have to do instead is do what's called a pattern match.

0:23:11.470 --> 0:23:13.780  
Dylan Bevan  
And this is where we say match the address.

0:23:14.640 --> 0:23:44.650  
Dylan Bevan  
With if it's none, then we just say there's no current address. If there's some, then we capture the address in the variable a, and then we can say print BA St so this completely stops you from having any null reference errors in your code, because normally in C#, if we just took an address and it could be nullable, people forget to put the null code check in. All your code has null checks all over the place and stuff to become a little bit trickier to read. You don't know if someone's previously done the null check, all that kind of stuff that you guys are used to.

0:23:45.570 --> 0:23:46.830  
Dylan Bevan  
Starts to be a pain in the \*\*\*\*.

0:23:48.750 --> 0:23:54.760  
Dylan Bevan  
So this is a kind of interesting thing. This is also coming to see Sharp. I think it might actually already be in there.

0:23:55.630 --> 0:23:57.740  
Dylan Bevan  
Umm, but it looks a little weird.

0:23:58.910 --> 0:24:1.170  
Dylan Bevan  
In my humble opinion, as they say.

0:24:2.820 --> 0:24:3.490  
Dylan Bevan  
So it's tight.

0:24:4.560 --> 0:24:8.390  
Dylan Bevan  
This is a options what we call it discriminated Union and so we can.

0:24:9.420 --> 0:24:17.210  
Dylan Bevan  
Declare discriminated union like this, so these are some fancy animals. So a giraffe is obviously fancy an elephant and a handsome dog is fancy animal.

0:24:18.500 --> 0:24:19.730  
Dylan Bevan  
So these are just like.

0:24:21.110 --> 0:24:27.160  
Dylan Bevan  
Empty, empty types. There's nothing to them, but this one has a string which I've called name here.

0:24:28.150 --> 0:24:31.250  
Dylan Bevan  
So these two will be nothing. This will have a value attached to it.

0:24:32.310 --> 0:24:35.750  
Dylan Bevan  
So I can also make some not so fancy animals.

0:24:37.90 --> 0:24:41.730  
Dylan Bevan  
Which are rats and bird eating spiders? I think pretty much everyone would agree with that.

0:24:42.710 --> 0:24:48.500  
Dylan Bevan  
And then we can have an enclosing discriminated union which takes to see their fancy or not fancy.

0:24:49.460 --> 0:25:2.360  
Dylan Bevan  
So this is like that terrible object oriented example that people usually teach where you have your animal based class and then you have a cat and a dog, which makes no sense because that's not really what object oriented programs all about. But this is how you can get like.

0:25:3.70 --> 0:25:6.390  
Dylan Bevan  
That uh sort of similar type hierarchy. If you wanted to do it.

0:25:7.860 --> 0:25:12.80  
Dylan Bevan  
So let's figure out now we want to describe the type of animal that we've got.

0:25:12.750 --> 0:25:27.610  
Dylan Bevan  
So we're just going to use pattern matching here, so we write our function describe animal which takes animal. Again, there's no types here at all, right? It's able to figure it out because I say match the animal with and as soon as I type, not fancy, it says. OK, this is the type of animal.

0:25:29.180 --> 0:25:36.750  
Dylan Bevan  
So if it's not fancy, I don't care about the value, so under score is throw away just like it is in C#. And then I'm just gonna return me.

0:25:37.510 --> 0:25:38.690  
Dylan Bevan  
If it's fancy.

0:25:39.620 --> 0:25:47.20  
Dylan Bevan  
Then I can do another pattern matching here, like a nested one, and so OK let's match F if it's a handsome dog, then I'm gonna get its name.

0:25:47.870 --> 0:25:52.290  
Dylan Bevan  
And so I can say, oh, the name is a fancy dog. Otherwise I'll just say it's fancy.

0:25:53.30 --> 0:26:6.220  
Dylan Bevan  
And then I take the result of that. This is the pipe forward operator. So this is gonna return me a strength and I can't return anything else here 'cause it will complain and say hey, everything else is returning a string in your returning an int.

0:26:7.510 --> 0:26:11.500  
Dylan Bevan  
And then I'm gonna pass that into the the printer function. So let's do that.

0:26:12.850 --> 0:26:19.520  
Dylan Bevan  
And then I'm gonna copy this one. So first thing is we're gonna describe A1. I can just pop it in here.

0:26:20.640 --> 0:26:20.930  
Dylan Bevan  
Right.

0:26:23.680 --> 0:26:26.280  
Dylan Bevan  
Describe animal. Oh, I forgot my semicolons again.

0:26:27.10 --> 0:26:30.700  
Dylan Bevan  
So the rat is not fancy, so you can see it did the men.

0:26:31.960 --> 0:26:32.750  
Dylan Bevan  
Meanwhile.

0:26:33.750 --> 0:26:34.700  
Dylan Bevan  
I'll put my.

0:26:35.360 --> 0:26:36.840  
Dylan Bevan  
Very own dog's name in here.

0:26:38.190 --> 0:26:40.140  
Dylan Bevan  
And then you can see Alisa fancy dog.

0:26:44.100 --> 0:26:44.670  
Dylan Bevan  
So.

0:26:47.410 --> 0:26:49.320  
Dylan Bevan  
Discriminated unions.

0:26:52.210 --> 0:26:56.480  
Dylan Bevan  
Can make things a lot easier, so let's see.

0:26:57.200 --> 0:26:59.590  
Dylan Bevan  
Now it can do that in this if this is gonna.

0:27:2.80 --> 0:27:6.970  
Dylan Bevan  
So we can have a database result here of a. So A is our generic type.

0:27:8.280 --> 0:27:17.910  
Dylan Bevan  
So see there an invalid ID. Whatever you gave us doesn't exist in the database, or we've got an error of EXN. So EXN is NFC F# shorthand for exception.

0:27:18.840 --> 0:27:21.530  
Dylan Bevan  
And then otherwise you have a success of.

0:27:22.640 --> 0:27:24.450  
Dylan Bevan  
Whatever the type was, right?

0:27:26.550 --> 0:27:29.640  
Dylan Bevan  
So now you can start compositing these things together.

0:27:31.120 --> 0:27:31.850  
Dylan Bevan  
So let's.

0:27:32.690 --> 0:27:45.60  
Dylan Bevan  
Stop making some functions so we can say you get data ID. We're gonna match if it's a null or white space ID then we return invalid ID otherwise we're gonna return success 100.

0:27:46.310 --> 0:28:16.410  
Dylan Bevan  
So these are very simple things. Obviously you could do your matching with this. Now we're gonna come to to function chaining in a little bit, but you can you can kind of wrap these things together. So maybe your check for this will return some or none and then if it's none, you're gonna basically eject immediately and just say it's a invalid thingy, or you're gonna carry on and go through all the functions. So you write these small functions that are very focused on specific jobs, and then you train them all together.

0:28:16.810 --> 0:28:18.280  
Dylan Bevan  
Into a bigger operation.

0:28:20.680 --> 0:28:23.350  
Dylan Bevan  
Does that make sense? So it's not just weird and coherent waffling.

0:28:27.230 --> 0:28:28.140  
Conf Room STUDIO C/3038 (14)  
Maybe you repeat that.

0:28:29.140 --> 0:28:33.800  
Dylan Bevan  
Yeah. Yeah. Let me see if I can do a a better, better way of doing this.

0:28:36.270 --> 0:28:41.50  
Dylan Bevan  
It's tricky because I don't wanna talk about workflow expressions just yet, but let's let's say we've got.

0:28:41.620 --> 0:28:42.70  
Dylan Bevan  
Umm.

0:28:43.410 --> 0:28:44.360  
Dylan Bevan  
And.

0:28:46.790 --> 0:28:52.620  
Dylan Bevan  
OK, so let's say let's finally ID ID equals and then we're just gonna copy this.

0:28:59.950 --> 0:29:4.880  
Dylan Bevan  
Umm, so if it's true, we're gonna return false, which seems stupid.

0:29:5.850 --> 0:29:12.170  
Dylan Bevan  
If it's false, are gonna return true, right? This is a stupid don't ever write code that looks like this is.

0:29:13.180 --> 0:29:17.190  
Dylan Bevan  
OK. But basically we have our valid ID, so if I.

0:29:17.970 --> 0:29:21.90  
Dylan Bevan  
Run that in here and then I type valid ID.

0:29:22.280 --> 0:29:22.750  
Dylan Bevan  
Umm.

0:29:23.780 --> 0:29:25.180  
Dylan Bevan  
Valid ideas? False, right?

0:29:26.370 --> 0:29:40.220  
Dylan Bevan  
So I can start wrapping this stuff into a pipeline. So let's think of the next thing. If we had a valid ID, maybe we wanna check the length is. Let me validate.

0:29:41.40 --> 0:29:42.270  
Dylan Bevan  
I didn't format.

0:29:43.20 --> 0:29:54.630  
Dylan Bevan  
I equals and let's just say let's like you can just do it easier and just say ID dot length. Let's make sure it's just strange, but no opportunity to figure it out just yet.

0:29:56.960 --> 0:30:0.250  
Dylan Bevan  
That might need all length is greater than three.

0:30:1.660 --> 0:30:12.990  
Dylan Bevan  
We don't need to do that. I'm not an interactive, so here we've got two function track. The first one is that our ID can't be a white space and the second one is that it has to be greater than three characters long.

0:30:14.470 --> 0:30:15.40  
Dylan Bevan  
Umm.

0:30:15.880 --> 0:30:24.670  
Dylan Bevan  
So what we can do is chain these together, so let's call it. Let validate ID functions equals and then I can say valid ID.

0:30:25.660 --> 0:30:27.850  
Dylan Bevan  
Well, let's just say match.

0:30:29.210 --> 0:30:30.440  
Dylan Bevan  
With that match.

0:30:31.990 --> 0:30:32.530  
Dylan Bevan  
Changes.

0:30:34.730 --> 0:30:37.250  
Dylan Bevan  
Not valid ID.

0:30:37.950 --> 0:30:39.80  
Dylan Bevan  
ID yes.

0:30:41.690 --> 0:30:43.40  
Dylan Bevan  
Now, if this is true.

0:30:44.410 --> 0:30:47.540  
Dylan Bevan  
So it is a valid ID. Then we're gonna return.

0:30:48.410 --> 0:30:49.30  
Dylan Bevan  
Match.

0:30:49.740 --> 0:30:51.870  
Dylan Bevan  
Validate ID format with.

0:30:55.250 --> 0:30:59.80  
Dylan Bevan  
This is true. Then we're gonna turn some of ID.

0:31:0.710 --> 0:31:2.620  
Dylan Bevan  
Otherwise within return none.

0:31:4.0 --> 0:31:8.70  
Dylan Bevan  
And then back here, we're gonna say whatever else, we're gonna send money.

0:31:12.650 --> 0:31:14.610  
Dylan Bevan  
To help client passing the argument that there we go.

0:31:16.110 --> 0:31:27.840  
Dylan Bevan  
So this is like a crappy way of chaining things together, but now you can see we just have these very short functions that are very easy to test on their own, and then you can chain them all together into a much bigger one.

0:31:29.880 --> 0:31:35.620  
Dylan Bevan  
Now if I was smarter about the return types, you can you can actually chain things together, so let's say.

0:31:36.390 --> 0:31:37.10  
Dylan Bevan  
Right.

0:31:37.440 --> 0:31:39.230  
Dylan Bevan  
And add 10.

0:31:40.490 --> 0:31:42.440  
Dylan Bevan  
X = X + 10.

0:31:43.750 --> 0:31:47.980  
Dylan Bevan  
No, that's not twenty X = X + 20.

0:31:48.990 --> 0:31:50.920  
Dylan Bevan  
We can now say like add 30.

0:31:53.60 --> 0:31:56.540  
Dylan Bevan  
We close at 10 and then we're gonna do this weird thing.

0:31:57.770 --> 0:31:58.680  
Dylan Bevan  
Not 20.

0:32:0.330 --> 0:32:7.260  
Dylan Bevan  
So what this basically has declared is another function, so this is a function that takes an integer X and it will return and it an integer back.

0:32:8.120 --> 0:32:20.930  
Dylan Bevan  
20 is an X that will, sorry is an integer that will return an integer again and then add 30. We're gonna just combine these two functions. So this is the equivalent of of saying this in C# at 20.

0:32:21.860 --> 0:32:32.660  
Dylan Bevan  
And and then we're gonna give it whatever our number is, and then we're gonna call add 10 with the. Actually, that's crap. That's, let's say that result equals had 10.

0:32:34.60 --> 0:32:39.270  
Dylan Bevan  
With the value that came in and then I'll say return and add 20.

0:32:39.670 --> 0:32:42.80  
Dylan Bevan  
Not telling so hard not result.

0:32:44.440 --> 0:32:45.890  
Dylan Bevan  
Right. So effectively this is.

0:32:47.300 --> 0:32:55.350  
Dylan Bevan  
Take whatever argument you get given here, pass it into a D10. Whatever the result of that is, pass it into add 20, and then return the result of that.

0:32:57.410 --> 0:33:1.600  
Dylan Bevan  
So you can use this if you match your types up nicely.

0:33:3.70 --> 0:33:5.760  
Dylan Bevan  
This makes it very easy to.

0:33:7.510 --> 0:33:8.980  
Dylan Bevan  
Composite functions together.

0:33:10.300 --> 0:33:24.650  
Dylan Bevan  
So the whole idea was functional programming because functions are first class citizens. That means you can pass a function around just like you would a variable. You write these very small discrete things and then you figure out how to fix them, put them together.

0:33:27.680 --> 0:33:42.890  
Dylan Bevan  
And this is really, I think, the biggest strength in F# is that it eliminates the entire stupid naming problem like you don't have managers or services or any of those weirdo classes that are just amalgamations of bits of pieces of functionality because you just declare your functions.

0:33:43.570 --> 0:33:44.0  
Conf Room STUDIO C/3038 (14)  
So.

0:33:43.740 --> 0:33:49.940  
Dylan Bevan  
Right, so far, the only class we've declared is is basically this, which doesn't have any functions as a data structure.

0:33:55.440 --> 0:34:0.870  
Dylan Bevan  
It did. Oh, you can declare an alias anytime, so I can say let.

0:34:0.960 --> 0:34:4.350  
Dylan Bevan  
Umm but like 18 equals at 30.

0:34:5.180 --> 0:34:6.510  
Dylan Bevan  
Right now I can just do.

0:34:7.560 --> 0:34:8.390  
Dylan Bevan  
Same thing again.

0:34:10.320 --> 0:34:14.850  
Dylan Bevan  
So again, sometimes this can make readability good. Sometimes this can make readability poor.

0:34:16.0 --> 0:34:27.170  
Dylan Bevan  
It will depends on how you want to do it, but this is very handy if you're using a lot of regular .net methods that have these crazy long names, you can just abbreviate them to whatever makes sense for you at the time.

0:34:30.660 --> 0:34:32.90  
Dylan Bevan  
Good deal.

0:34:34.590 --> 0:34:43.640  
Dylan Bevan  
OK, we're gonna look at workflow expressions. So these are also called monads, which you probably all monads depending on how you wanna say it.

0:34:44.530 --> 0:34:45.150  
Dylan Bevan  
Umm.

0:34:45.950 --> 0:34:50.580  
Dylan Bevan  
Nothing too scary about it, so we'll go through it with the option type.

0:34:52.250 --> 0:34:58.520  
Dylan Bevan  
So Blair, wherever we have, where we had that option of none and some right. So instead of doing a null.

0:34:59.240 --> 0:35:10.210  
Dylan Bevan  
You either have a value or you don't have a value. So if you have lots of things that are put together in fact, let me just copy some string, some code over that I've got here.

0:35:12.290 --> 0:35:13.480  
Dylan Bevan  
Dear.

0:35:16.970 --> 0:35:24.720  
Dylan Bevan  
Yeah, this is showing some tedious code so validates string which is just that null or white space which is gonna return none or some.

0:35:25.470 --> 0:35:28.440  
Dylan Bevan  
And I've abbreviated it to the S and then we take our address in.

0:35:29.440 --> 0:35:33.810  
Dylan Bevan  
And we say match the address street with if it's.

0:35:35.220 --> 0:35:49.250  
Dylan Bevan  
Yeah, if it's. If the street is empty then it's none. Otherwise, if the address is true, then we'll check St two and if St 2's OK, then we'll check Street City and this starts to build up this horrific mess of like nested if statements effectively.

0:35:50.330 --> 0:36:5.400  
Dylan Bevan  
Umm, but we can. We can get around this with something called monads, so this is effectively how async await works in C#, but in F# you can extend that model to whatever you wanna do. So bear with me as long as this part makes sense so far.

0:36:6.360 --> 0:36:7.260  
Dylan Bevan  
We should be bingo.

0:36:8.640 --> 0:36:14.110  
Dylan Bevan  
So first thing we're going to do is to create a type of function.

0:36:14.980 --> 0:36:27.970  
Dylan Bevan  
That takes F, right? So F is a function. So like I said, in F# you can pass functions just like you would variables O we declare this function execute on sum which takes a function and an option.

0:36:28.820 --> 0:36:34.20  
Dylan Bevan  
And then it'll say match the option type that I've got with. If there's some and there's a value in it.

0:36:34.750 --> 0:36:38.360  
Dylan Bevan  
Pass that value into the function F, otherwise return none.

0:36:39.510 --> 0:36:47.160  
Dylan Bevan  
Now, because we were returning none here, F must return an option type, so it must return some or none. OK.

0:36:49.520 --> 0:36:56.630  
Dylan Bevan  
Now we're gonna do something which is typically called a builder, and in this example it's almost always called a maybe builder.

0:36:58.220 --> 0:37:9.860  
Dylan Bevan  
So it has some specific method names. There's a lot more than this. You can go through if you want to build a full monad, but in this case we're just looking at this one. So we'll say when when you get asked to bind.

0:37:10.990 --> 0:37:18.320  
Dylan Bevan  
And you got given your value and your function then we call our execute on some function with F and the value.

0:37:19.30 --> 0:37:25.310  
Dylan Bevan  
When it comes time to return, then we're gonna return the option of X. This is just to wrap it up, OK?

0:37:26.0 --> 0:37:29.170  
Dylan Bevan  
This probably all seems super weird, but in a second I'll make sense.

0:37:30.870 --> 0:37:36.0  
Dylan Bevan  
So now I declare an instance of that type. So this is a class. This is its constructor and should have gone over that.

0:37:36.860 --> 0:37:41.470  
Dylan Bevan  
Umm, so we're going to declare an instance of this called maybe, which is a new maybe builder.

0:37:42.260 --> 0:37:43.940  
Conf Room STUDIO C/3038 (14)  
Wait, I thought you said no classes.

0:37:44.490 --> 0:37:51.760  
Dylan Bevan  
Yeah, well, this is like a weird one, but it does give us a lot of power. So this is the magic on this one is, I'm gonna say.

0:37:54.80 --> 0:38:11.670  
Dylan Bevan  
Is it a valid address? So we're gonna declare our function here. We're gonna take an address, and then we write, maybe, and then we give it our scope. And within here, every time I write, let exclamation marks, we call it, let bang. So everywhere else we do let bang is like let's.

0:38:13.170 --> 0:38:14.270  
Dylan Bevan  
Unwrapped the type.

0:38:15.70 --> 0:38:27.390  
Dylan Bevan  
I don't really care what the value is, but I'm saying address St 2 right? So VS remembers my stupid alias for validate strings, so we're doing validate strings St St, two, City, country post code, blah blah blah and then finally we return the address.

0:38:29.200 --> 0:38:29.670  
Dylan Bevan  
So.

0:38:30.990 --> 0:38:33.480  
Dylan Bevan  
If I if I put X in here the example.

0:38:34.230 --> 0:38:53.340  
Dylan Bevan  
When I hover over this, it'll tell me that the value of X is string because it's unpacked the option type. So we're not dealing with option types here, we're just dealing with the actual values. In fact, nowhere in here do you see an option type right? We're just literally accessing the values in here and our address that we return is an address, not an option type.

0:38:54.600 --> 0:38:58.600  
Dylan Bevan  
But if I look at what the type of is, valid address is.

0:38:59.610 --> 0:39:3.0  
Dylan Bevan  
You might not be able to see it, but it takes an address and it returns an address option.

0:39:3.910 --> 0:39:7.260  
Dylan Bevan  
Right, so now I can say let the address.

0:39:8.140 --> 0:39:15.750  
Dylan Bevan  
Equals is valid address and then I'm gonna give it my. What did I call it? Like something fancy up here. Sample address.

0:39:20.50 --> 0:39:24.330  
Dylan Bevan  
And let's hope I copied that into interactive, so I'm just gonna run all of these through.

0:39:26.220 --> 0:39:27.280  
Dylan Bevan  
Interactive.

0:39:28.750 --> 0:39:46.330  
Dylan Bevan  
You know, on this year and hopefully we got ohh it's got an option of none. So something right we failed validation here. Now this is hopeless cause it doesn't actually tell us what the validation is. But the key point is you can make this operate on any type that you like. So if you want to do async in F.

0:39:48.180 --> 0:40:1.410  
Dylan Bevan  
It's super easy to do. You basically would just type async in here async and then you have your async function in here so you can let bang response equals HTTP Web request.

0:40:2.560 --> 0:40:11.290  
Dylan Bevan  
Web Sender, whatever those things are, blah, blah, blah blah blah and then pass in your value and what you get back is your response and all of this is asynchronous and you don't have to do the await nonsense.

0:40:12.430 --> 0:40:15.560  
Dylan Bevan  
Alright, so this is where that syntax came from because.

0:40:17.400 --> 0:40:36.260  
Dylan Bevan  
That in that shop, effectively the the builder that they using rats up task of tea, which is what you use in C#, right? You have to pass task of tea around. It's exactly the same thing here except with if it doesn't have to be a task you can use whatever you like in there to get going. So you can really simplify your code and make it very easy.

0:40:38.720 --> 0:40:44.590  
Dylan Bevan  
That's pretty. That's pretty wild. So if if you're not fully with that, no worries.

0:40:48.360 --> 0:40:56.70  
Dylan Bevan  
I got one last weird thing and then I'll I'll finish with the magic. Like, make the Statue of Liberty disappear. Magic trick.

0:40:57.890 --> 0:40:59.490  
Dylan Bevan  
So for concurrency.

0:41:1.140 --> 0:41:17.730  
Dylan Bevan  
It is simpler that easy enough shop. Obviously everything immutable, right? So you know that you're not gonna have any of those weird race conditions on threads and stuff. But I'm gonna declare this inbox message so we have increment, decrement and get which has this weird async reply channel.

0:41:18.640 --> 0:41:22.300  
Dylan Bevan  
So this is just a discriminated union. There's nothing particularly fancy about that.

0:41:23.980 --> 0:41:25.510  
Dylan Bevan  
And then I'm going to declare.

0:41:26.570 --> 0:41:29.930  
Dylan Bevan  
Our mailbox processor, so this is a built in F# type.

0:41:31.40 --> 0:41:36.270  
Dylan Bevan  
And I'm going to call it an agent, so if anyone's done active based programming, this is effectively your actor.

0:41:36.950 --> 0:41:41.400  
Dylan Bevan  
So we create a mailbox processor, it's type, it's generic type is inbox message.

0:41:41.990 --> 0:41:57.340  
Dylan Bevan  
And then we call start and we call our inbox and then we have a recursive function in here. So F# is tail recursive by the way. So you can as long as you know how to write to our recursive function. You can run this thing forever and it will won't blow the stack.

0:41:58.590 --> 0:41:59.30  
Dylan Bevan  
Uh.

0:41:59.770 --> 0:42:4.440  
Dylan Bevan  
So this function is called loop. It takes N which in this case is an integer.

0:42:5.310 --> 0:42:11.940  
Dylan Bevan  
And they can figure that out because I'm passing it or calling it down here with zero. And then this is that lovely async workflow builder, right?

0:42:12.620 --> 0:42:28.40  
Dylan Bevan  
So we're saying that message equals Inbox received. So now with your knowledge of task from C#, you know this is basically a non blocking call. So we can just spin up a billion of these things and then they'll just hang around waiting for a message to be posted to them.

0:42:29.270 --> 0:42:44.50  
Dylan Bevan  
Once we've got our message, we'll say match that message with. If it's increment, then we'll take N that was given to us and add one to it. If it's decrement and minus one, or if it was get, we will reply to that Channel with the number that we currently have.

0:42:44.970 --> 0:42:57.40  
Dylan Bevan  
And we assign whatever the value we return here is. It's either n + 1 and minus one or N we assign that to M and then we call back this function again, where N is now M and then we wait for the next message to come in.

0:42:59.750 --> 0:43:4.380  
Dylan Bevan  
Cool. So now let's put some of this stuff together that we've done.

0:43:9.650 --> 0:43:11.680  
Dylan Bevan  
So from excuse me.

0:43:12.540 --> 0:43:15.430  
Dylan Bevan  
One to 100 and $0.99 an easy array expansion.

0:43:19.590 --> 0:43:23.900  
Dylan Bevan  
Excuse me? And this is basically equivalent to select. Good grief.

0:43:25.230 --> 0:43:26.440  
Dylan Bevan  
So given our X.

0:43:27.410 --> 0:43:29.780  
Dylan Bevan  
We're gonna match X with if it's.

0:43:31.160 --> 0:43:33.110  
Dylan Bevan  
When we do modulo 2, if it's 0.

0:43:34.0 --> 0:43:39.630  
Dylan Bevan  
Then blah blah blah, we can print something. Otherwise I'm gonna post increment to our agent. Otherwise I'm gonna post decrement.

0:43:40.700 --> 0:43:46.690  
Dylan Bevan  
And I'm gonna run this in parallel and then run synchronously, which basically means just wait here until you get the result back.

0:43:47.810 --> 0:43:49.500  
Dylan Bevan  
Finally, when we get the result back.

0:43:50.160 --> 0:44:1.260  
Dylan Bevan  
Which is we don't care about, it's not actually doing anything. We'll say post and reply. Get me the channel. And what's the current count of counter value? So I put this printf in here so.

0:44:2.10 --> 0:44:5.630  
Dylan Bevan  
If I were to do it, you could see well, why don't we just do it?

0:44:6.540 --> 0:44:9.270  
Dylan Bevan  
You can see that it's executing on on many different threads.

0:44:11.350 --> 0:44:15.340  
Dylan Bevan  
Well, it's not happy that this value is not a function can't be applied.

0:44:16.920 --> 0:44:18.50  
Dylan Bevan  
Dear.

0:44:21.350 --> 0:44:24.480  
Dylan Bevan  
Not quite sure why it's moaning about. That's probably why I commented it out.

0:44:25.330 --> 0:44:27.660  
Dylan Bevan  
Oops, it's a different.

0:44:28.480 --> 0:44:30.40  
Dylan Bevan  
Different syntax on there so.

0:44:31.440 --> 0:44:32.930  
Dylan Bevan  
You just have to take my word for it.

0:44:34.500 --> 0:44:37.950  
Dylan Bevan  
Is that this is gonna run on a different thread, but I'm gonna post that in.

0:44:38.850 --> 0:44:43.20  
Dylan Bevan  
And it'll take a while. Ohh, agents not defined yet. 'cause I gotta do the rest of the stuff right?

0:44:46.940 --> 0:44:52.170  
Dylan Bevan  
And now this will happily do 199 operations and eventually we end up with minus one.

0:44:53.200 --> 0:44:56.650  
Dylan Bevan  
Uh, we should make sense because we started at 1, not 0, right?

0:44:58.190 --> 0:45:1.930  
Dylan Bevan  
Umm, so all of that is is how you do concurrency.

0:45:4.70 --> 0:45:12.870  
Dylan Bevan  
Alright, now I'll do the magic trick which I've been showing for forever, and C# still doesn't have any equivalent. So the first thing is I'm gonna.

0:45:13.820 --> 0:45:28.170  
Dylan Bevan  
Use a couple of references so with our sharp interactive you can just reference a nugget package like this. So I'm gonna have sharp dot data. I'm gonna pull this nugget package into my interactive feed.

0:45:33.610 --> 0:45:37.880  
Dylan Bevan  
And then I also want same thing.

0:45:39.410 --> 0:45:42.990  
Dylan Bevan  
I want to it's a different one here I want.

0:45:47.850 --> 0:45:49.390  
Dylan Bevan  
Let's plot dot.

0:45:50.210 --> 0:45:51.10  
Dylan Bevan  
Love me.

0:45:51.830 --> 0:45:52.810  
Dylan Bevan  
Not interactive.

0:45:55.570 --> 0:45:58.60  
Dylan Bevan  
OK. So we're gonna have a look at something called type providers.

0:45:59.60 --> 0:46:4.10  
Dylan Bevan  
Which is just still blows my mind as to how weirdly magic it is.

0:46:5.470 --> 0:46:9.730  
Dylan Bevan  
So what type providers will do is now go and build a.

0:46:11.100 --> 0:46:13.570  
Dylan Bevan  
Actually, let's copy it in here. Let me.

0:46:15.120 --> 0:46:18.20  
Dylan Bevan  
I'm just used to get the correct thingies.

0:46:19.190 --> 0:46:21.900  
Dylan Bevan  
With F# me type open instead of using.

0:46:23.170 --> 0:46:26.210  
Dylan Bevan  
So let's do you open a shop later and then open?

0:46:27.910 --> 0:46:31.140  
Dylan Bevan  
And it's not dot dot.

0:46:33.700 --> 0:46:34.470  
Dylan Bevan  
There we go. OK.

0:46:36.70 --> 0:46:37.940  
Dylan Bevan  
So I'm gonna copy this thing and.

0:46:39.370 --> 0:46:47.180  
Dylan Bevan  
Now if anyone wants to, you can just go browse any old website you want. Wikipedia is a good one and so I'm gonna bring this up.

0:46:49.10 --> 0:46:49.910  
Dylan Bevan  
And then.

0:46:51.470 --> 0:46:57.240  
Dylan Bevan  
Actually, I'm just going to post this in the chat and then you'll have to get there on your own cause it's pain to share stuff.

0:47:0.510 --> 0:47:1.380  
Dylan Bevan  
Whilst presenting.

0:47:6.500 --> 0:47:13.820  
Dylan Bevan  
Does the runtime convert it to an option for you? No, there's actually a noble type Dustin to answer your question about nobles, OK, so.

0:47:14.790 --> 0:47:18.600  
Dylan Bevan  
If you have a look at this web page, it's just a web page about Microsoft Windows.

0:47:19.460 --> 0:47:23.140  
Dylan Bevan  
And if you scroll about halfway down, I'm just going to pull it up myself.

0:47:24.600 --> 0:47:27.290  
Dylan Bevan  
There is a table on that page that shows you.

0:47:29.90 --> 0:47:31.780  
Dylan Bevan  
The value the number of usages.

0:47:33.410 --> 0:47:34.60  
Dylan Bevan  
Umm.

0:47:35.180 --> 0:47:37.240  
Dylan Bevan  
Well, all the versions of Windows.

0:47:39.360 --> 0:47:41.300  
Dylan Bevan  
So I'm going to.

0:47:44.450 --> 0:47:50.70  
Dylan Bevan  
This basically goes and has a look at that page and then it tries to build types based on that page.

0:47:51.610 --> 0:47:55.440  
Dylan Bevan  
Which sounds bonkers, but now I can say.

0:47:56.940 --> 0:47:59.190  
Dylan Bevan  
Now that you've declared the types, you know what they are.

0:48:0.620 --> 0:48:5.870  
Dylan Bevan  
Go and load those into a data structure which I'm gonna call Windows data.

0:48:8.610 --> 0:48:13.720  
Dylan Bevan  
Now, I don't think I get Intellisense in the interactive down here. I don't.

0:48:14.400 --> 0:48:19.320  
Dylan Bevan  
That's how you what that me. Let me pull these things up the top.

0:48:20.380 --> 0:48:23.490  
Dylan Bevan  
Upload these things in here so you can see just now.

0:48:24.200 --> 0:48:25.830  
Dylan Bevan  
Crazy. This is in real life.

0:48:27.870 --> 0:48:32.980  
Dylan Bevan  
And then I come down, right. And I'm gonna paste that same thingy that I just did there.

0:48:34.90 --> 0:48:35.200  
Dylan Bevan  
While blah blah.

0:48:38.0 --> 0:48:41.20  
Dylan Bevan  
OK, so now if I do Windows data dot.

0:48:42.470 --> 0:48:45.140  
Dylan Bevan  
How are you serious? Outside? I didn't close that off earlier.

0:48:46.60 --> 0:48:48.200  
Dylan Bevan  
Got another error up here. Let me just get rid of that.

0:48:49.240 --> 0:48:51.310  
Dylan Bevan  
From there now, hopefully it'll do this thing for me.

0:48:52.590 --> 0:49:5.210  
Dylan Bevan  
OK, so now I've got the raw HTML and the type provider is also found some tables and some lists so I can do tables and dot and this is gonna give me the tables that are defined on that page.

0:49:6.610 --> 0:49:15.20  
Dylan Bevan  
Right. So there's one called Windows and then there's also one helpfully called table three and table 4I happen to know the table four. It's gonna have a look at that, and I'm gonna take the rows.

0:49:15.720 --> 0:49:16.560  
Dylan Bevan  
But that table.

0:49:17.510 --> 0:49:26.190  
Dylan Bevan  
And then I'm gonna pass those. I think they're in array. Let's see. They are an array. OK, so I'm gonna do an array dot fitter, which is basically a four each.

0:49:26.860 --> 0:49:30.590  
Dylan Bevan  
And then a function which given rho.

0:49:32.110 --> 0:49:46.380  
Dylan Bevan  
We'll say row, dot and now you can see it's actually figured out. Hey, there's desktop OS and stat counter in there, which if you have a look at the web page, if you're following along, you should see those are the column headers that are in there.

0:49:47.290 --> 0:49:55.160  
Dylan Bevan  
Now, if you don't believe me, you can give me any old link to Wikipedia. It's long as it's safe for work, which I think most of Wikipedia is. You can paste it in and we'll go figure out the types of that.

0:49:56.70 --> 0:49:57.630  
Dylan Bevan  
Umm, but now what I can do?

0:49:58.740 --> 0:50:2.170  
Dylan Bevan  
Is I can go and say, actually let's do a map on those things.

0:50:3.230 --> 0:50:8.260  
Dylan Bevan  
They were, so we're going to do an array dot map because we're gonna pull these things out. So given a row.

0:50:9.250 --> 0:50:10.520  
Dylan Bevan  
I'm going to return a tuple.

0:50:11.630 --> 0:50:15.630  
Dylan Bevan  
And I'm gonna return a temple of the desktop OS.

0:50:16.340 --> 0:50:18.480  
Dylan Bevan  
And then the stack counter.

0:50:21.870 --> 0:50:28.30  
Dylan Bevan  
Now what I can do using the magic of the Plotly one that I had as well as I can say chart dot pie.

0:50:29.820 --> 0:50:33.10  
Dylan Bevan  
And then let's do a chart with title.

0:50:34.990 --> 0:50:37.110  
Dylan Bevan  
Our windows message.

0:50:39.220 --> 0:50:44.430  
Dylan Bevan  
Now I'm gonna pipe that into a chart with the legends because we wanna know what we're looking at.

0:50:46.180 --> 0:50:48.880  
Dylan Bevan  
And then lastly, I'm gonna do chart that show.

0:50:50.910 --> 0:50:56.640  
Dylan Bevan  
OK, so I have all those things down now. What's the last thing I did? Windows data. OK, so now I'm gonna pipe this into.

0:50:57.450 --> 0:50:57.980  
Dylan Bevan  
I.

0:50:59.110 --> 0:51:0.240  
Dylan Bevan  
Thing in the jigani.

0:51:1.710 --> 0:51:4.740  
Dylan Bevan  
Unexpected symbol, blah blah blah.

0:51:6.450 --> 0:51:11.210  
Dylan Bevan  
Yeah. What's it not happy with you? I missed a closing statement. There we go. Let's try that one again.

0:51:12.970 --> 0:51:13.460  
Dylan Bevan  
OK.

0:51:14.440 --> 0:51:17.270  
Dylan Bevan  
And you can't see it, so I'll have to share my screen.

0:51:18.260 --> 0:51:19.40  
Dylan Bevan  
After all that.

0:51:21.90 --> 0:51:21.980  
Dylan Bevan  
Share.

0:51:23.300 --> 0:51:25.640  
Dylan Bevan  
My browser, Umm. But this is the result of it.

0:51:26.740 --> 0:51:30.610  
Dylan Bevan  
It's actually now I can show you the Wikipedia page, so this was the table we were actually looking at.

0:51:31.900 --> 0:51:34.830  
Dylan Bevan  
And this is the data that was provided from it.

0:51:35.580 --> 0:51:41.390  
Dylan Bevan  
And then you can click through these things and just say how does Windows 7 compared to Windows 11?

0:51:42.460 --> 0:51:42.740  
Dylan Bevan  
Right.

0:51:43.740 --> 0:51:47.650  
Michael Mashayekh  
So you just got Intellisense for scraping the web page.

0:51:48.60 --> 0:51:48.570  
Dylan Bevan  
Yes.

0:51:49.440 --> 0:51:50.70  
Michael Mashayekh  
That's amazing.

0:51:49.510 --> 0:51:57.340  
Dylan Bevan  
And there are there are type providers for everything. So for SQL there's type providers, there's type providers for Jason, so.

0:51:58.130 --> 0:52:1.860  
Dylan Bevan  
I think we've all been there where you have some thing you're trying to.

0:52:3.860 --> 0:52:27.130  
Dylan Bevan  
Interrupt with and you get this huge stack of Jason back. You can actually just run that through a type provider and then it will give you a concrete type for it. You don't have to manually go and convert it to a type or find some online operators to do it. There type providers for swagger endpoints right? So if you are consuming data from another endpoint, you don't need to go get their libraries and blah blah blah. You can just point the type provider at it and then it will generate all the types for you.

0:52:30.550 --> 0:52:46.480  
Dylan Bevan  
So that's the crazy thing. There is still no, no C# equivalent on that. So if you're like, well, you know, it's cool, but it's just another language. I think there's a reason that C# hasn't done it, and that's because it's just way easier to work with functions.

0:52:47.270 --> 0:52:54.630  
Dylan Bevan  
So my my last thing I would implore is that you go look at a functional language and just spend a week.

0:52:55.320 --> 0:53:7.260  
Dylan Bevan  
Just looking at it, trying it out and your universal change and I've sharp is like the easy on ramp drug because you already know all the all the net BCL. So you can pretty much do anything with it.

0:53:9.870 --> 0:53:16.120  
Dylan Bevan  
All right, we got 2 minutes for questions. If there's any questions or I should go through the chat really and see if there's any questions in there.

0:53:16.840 --> 0:53:20.950  
Dylan Bevan  
There currently is audio function, first class and it's messaging functions take multiple arguments. Mm-hmm.

0:53:26.700 --> 0:53:33.270  
Dylan Bevan  
Yeah. So the dynamic it's not good at dynamic because it screws up with the type inference.

0:53:34.780 --> 0:53:47.310  
Dylan Bevan  
Umm, so usually if you need to do something like the classic one is. If you want to do any Excel data manipulations, there's a type provider for Excel, but if you don't have that then you write that portion in C# and then as a library and then you just consume that within F.

0:53:50.90 --> 0:54:8.800  
Dylan Bevan  
Math with the Switch case. So the switch case actually does guard operations as well. So if you are doing a case match on a numerical value for example, you can say match X when X is greater than 100 and do this specific thing, or if it's when it's less than this, do this specific thing.

0:54:9.680 --> 0:54:20.190  
Dylan Bevan  
Umm, there's active pattern matching which I didn't have the chance to go into, but that's when you can basically do a match where it's actually running a function under the hood and we'll do some plans and stuff in there. So it's very cool that way.

0:54:24.610 --> 0:54:28.970  
Dylan Bevan  
I think that's all the questions any in person questions or we wrap it up.

0:54:32.30 --> 0:54:35.360  
Dylan Bevan  
Sweet. Most people stayed awake, so thank you very much.

0:54:38.230 --> 0:54:38.450  
Conf Room STUDIO C/3038 (14)  
Thanks.

0:54:38.80 --> 0:54:46.130  
Dylan Bevan  
I will end the recording and I took up all the time, so no, no spare time for anybody. Apologies for that, OHG. My God, it's this start recording.

0:54:52.830 --> 0:54:53.160  
Conf Room STUDIO C/3038 (14)  
So.

0:54:49.690 --> 0:54:54.10  
Dylan Bevan  
So the whole thing wasn't recorded. No, it's still recording. OK, now I'm confused with teams.

0:54:56.730 --> 0:54:59.400  
Dylan Bevan  
Oh no, it says transcription. Stop transcription.

0:55:0.410 --> 0:55:0.840  
Dylan Bevan  
OK.

0:55:1.740 --> 0:55:11.890  
Dylan Bevan  
So this learning session is going to be a text description only which will be super fun for everybody. I'll I'll stick the source code in the in the link.

0:55:13.930 --> 0:55:14.390  
Dylan Bevan  
Awesome.

0:55:15.70 --> 0:55:15.880  
Dylan Bevan  
Thanks everybody.

0:55:16.290 --> 0:55:16.680  
Conf Room STUDIO C/3038 (14)  
Dylan.

0:55:16.860 --> 0:55:17.550  
Dylan Bevan  
The elevator.

0:55:23.820 --> 0:55:24.210  
Conf Room STUDIO C/3038 (14)  
Working.