ITEC 380—11

Radford

Steven Oake

September 14, 2015

Karasch and Associates

Captioned by: Adina King

"This text is being provided in a rough draft format. Communication Access Realtime Translation (CART) is provided in order to facilitate communication accessibility and may not be a totally verbatim record of the class."

 (Call started at 5:45)

» INSTRUCTOR:  Hello.  I'm going to take off my shoes.  I am taking my shoes off right now.  I won't stand up so you guys won't ever know.  I guess for Chris his power is off.  He's taking the bus down to the library to join the class.  So he will be a little bit late joining us.

I am full of energy.  I was stretching not yawning.  I was in Floyd.  Which night was that?  They run together -- Saturday night I guess.  Maybe it was Friday night.

Yeah, Friday night.

The -- I forget what it's called.  I think it was a coffee shop but actually a dog town -- yeah.  I think my wife went in there and was listening to the band more.  I was waiting outside for her.  Yeah, it would have been interesting.

We're getting a few more people connecting here.  I'll start the sharing and I'll start the recording.  I guess I have to start the recording first because sharing will cover up the screen.

Hi, Shawn.  Starting the recording.  Bring up the window.  I have this here -- if you go -- if you go to the lecture's page and click on the link I think it's call -- it should be the last lecture -- we just barely got to that material.  It looks like gook -- I'll fix that but for now you can download the file and then -- inside for (inaudible) -- it has a lot of the stuff that I will be flushing out in a good -- not just sit there and see the answers as I'm talking about did -- it's good to be thinking about it.  We're explaining why we're getting the answer and building it up but don't feel you have to get every last bit down.

Thank you.  You should see what I'm typing after I share my screen rather than just shortening to share my screen.

Close various windows here.

Okay.  This is the message that I just spoke out.  I actually put the file up.  I found out today it was looking weird.  So if you tried it last week it would have looked weird but open it and it looks fine.  If it has any graphical files in it that's what will happen -- it shouldn't have the graphical elements in it.  That was a typo on my part.

So I want to start talking -- so first of all, I think I've gotten -- I haven't checked e—mail super recently but I think only one student has submitted -- are you getting it to me -- I think I called it quiz one.  And I realizes you may not be able to do as well as you hope -- hopefully you get something there working.  I put a time limit on it so it doesn't chew up your time.

Something I will show by the way, I'll type it in here -- ignore the row column part of that -- it was for students in class and you all got -- if you got on the grade shape you got the one point quiz -- they all got one point except for the few that goofed it up.

If I -- where was I?  I was going to open the browser and do something in the browser.

If you want practice --

 (LAUGHTER)  -- if you want practice writing small Java programs, really I mean functions, the biggest thing I'm after is decomposing problem in functions -- they get inputs and work with those inputs and work with those results.  Nothing to do with scanning or printing.  They return a result.  You can look at Javabat.com -- we can look real quick right now --

Go to a browser -- Java bat.com -- you can set up an account and keep track of problems you solved and that kind of stuff.  But you can look up problems and go to Java, warm up one or warm up two.

Warm up two -- and here they have about 15 different problems, I'll just click on one of them.

It has a simple description.  You're given a string and a non—integer -- return N copies of that front of the string.  So (inaudible) -- and so on.

So now you're task is to go and write code.  The nice thing you don't need to do double static -- it's all the in the browser.  They give you the header here.  It takes a string and a INT -- now you can go ahead and figure out how to do that and I'll leave it -- you can go ahead and have this.  If your Java this week, this is a place to go -- send me an e—mail for help.  The one thing I like about coding bat is it really focuses on individual function is where I want to emphasizing the time we spent not on IO and keyboard -- it's just a distraction of the main work of solving a problem.

The thing I don't like about it is their problems are all kind of silly.  This one is not as bad as many of them.  The I in homework I try to have problems that are useful in real life for people end up writing.

If you can take at that quiz and send it to me that would be appreciated.

So okay, questions or thoughts about the homework that's currently out?

You have until Wednesday or so but you have the information that you need to work on that.

And I said you can go ahead and turn it in on I think -- I miss typed it, so turn it on Thursday.  Have people looked at that yet?

No one's willing to admit they looked at it?  Since we didn't have other stuff going on, hopefully this week you have a chance to look at these things and if you have questions you can ask.

And we'll play it by ear about turning in homework.  I can have things due the start of class and it's nice because we can talk about the solution exactly.  I can have things due on Tuesday or Wednesday, say Wednesday night, that way if you couldn't do something you can ask about it in class have some direction and have time on it but it slows down the pacing.  You'll be writing code nine days earlier so it's harder to stay on top.

Let me know preferences if you think things are going too fast or too slow.

Okay.  So we ended last week -- I talked real briefly about COND -- it stands for conditional and (inaudible) -- I gave one quick example -- I define 9 to be 17 -- if this condition is true then this will be the answer.  If this condition is true then I'll put the answer over here.

The third in this example -- if the third condition is true -- it looks like question, answer pairs in square brackets.

I'll find it and put it up.

From there most recent semester and then the lecture's link -- it's called using COND -- if you download it and start up Racket and have Racket open that file.  So we have question, answer, pairs, the question can be anything that ends up to be evaluating to Boolean.  You do a lot of arithmetic in there, whatever you want to do in there.  As it is when you ran this we got a Aloha.  Defined N to be 17 -- less than ten, no.  Is N less than 20, yes, go ahead and evaluate the answer for that case which is Aloha.

Did I mention some of the stuff that is highlighted in black -- did I mention that?  Can you infer what it might mean from what's covered in black?  It has nothing do to do with errors.  It should be this file.  There's not a whole lot in it.

It should look more or less like this -- there's a whole bunch of stuff that's commented out -- most of it is comment.

I'll make sure -- I'll fix that over our first break.

So one thing to know is -- any questions on that black part?  It's a code coverage tool.  It's telling you what lines actually were touched, were executed or evaluated.  Anything that's covered in black -- that code was never looked at it.  Don't assume it's working because we never run it.

It's not debug it's code coverage.

If you have interesting test gauge cases -- after you run all your test cases -- you better run every bit of your program -- if there's a little bit of program that never got looked at either they're not comprehensive or they're unreachable or should be fixed.

And I'll mention -- I'll change some of these here.  Take this little thing and take the co—sign of all that -- it's a silly thing to do -- I can run this -- I get some answer.  0.0666 -- that's whatever this expression was -- code sine N squared minus 99.  You have questions and answers and usually the questions are relatively simple like if the conditions reach are relatively seven and most of this can be complicated code -- this might span several lines.  We'll see examples of that, two or three lines.

Let's look at a place where we might use this.

I'm going to go through and look at a function called turned that go ahead -- it's a soft light controller.  You tell it what the stoplight is currently at, the color showing red or yellow or green and it will tell you what color should come up next.  Okay?

So first of all, the first thing to think about, if this is our problem, what data type should we use to represent our data?  The light is currently red or green how should we represent that -- what data type should we use?

Good -- several people chiming in.  Bullion enumeration -- I'll say (inaudible) -- once of the answers I like -- Racket doesn't have built in -- we're going to use something similar.  ID use 012 is for green, yellow red or something like that -- a numerator type is like months of the year or days of the week.  It can be Monday, Tuesday, Wednesday, up through Sunday and no more, no less.  You can say I'm going to use 0 for Sunday and 1 for Monday, and 6 for Saturday, that's fine too but I shouldn't take 7 times Friday.  I'm using INTs (inaudible) -- will let me write 7 timed Friday no problem.  The computer knows is it just a name or a label -- it doesn't let me multiply.  Even in Java there might be a number associate with the (inaudible) --

Okay.  Yeah, yesterday when we were talking about different languages I mentioned a list and Scheme and Racket and they're all kind of the same.  So Scheme and Racket are very close.  The Racket people didn't want to help to wait for the Scheme standard to be approved by three year committee process in order to add new features so they decided let's not call Scheme and piss somebody off let's add the features we want to add.  So Racket is basically Scheme.

So I want numerated type for soft lights.  Racket doesn't have it.  It has something similar to -- similar to strings -- I'll scroll down here and reveal some stuff.  I'm going to say a stoplight color, SLC -- that's my technical term -- the fact that I'm from Salt Lake City has nothing to do with this -- SLC is one the possible things -- quote red, quote yellow, quote flashing red, quote flashing yellow -- what's the tick mark mean?  It's a quote.  I can say tick mark red -- the answer says text mark red -- it's a symbol that valuates itself.  It looks like a string.  If I had a tick mark on both sides and say that's exactly a string like JavaScript or whatever.  The only difference between single symbols -- if you're taking substrings, upper, lower case conversions, you want to use a string.  If you want I have five different states I want to represent, I'm going to give it a tick mark -- I'm not going to do string pen -- the only two -- the only thing I can do with symbols is I have two different symbols I can ask whether or not they're equal.  That's the only thing a computer knows how to do with symbols.  They're kind of handy and I like using them in this situation.  They don't -- strings you expect -- they're waiting for me to use substring and I'm never going to do that with stoplight colors.

Stoplight color is either red, yellow, green, flashing red, flashing yellow -- I'm going to write a function and tell me what color is going to come up next.

So the first thing we do write our test cases.

So sure -- check—expect -- write me one test case -- on your computer paste if you want one of the test cases for this function term.  Reminding ourselves how to write --

So hash T -- more than that.  Hash T's not -- check—expect takes the actual output and then my desired output.

So strings string -- well not quite string -- so again -- this I expect you to know from last week and the stuff we've done desired to learn and so on, be able to put in check—expect.  The hard part -- we haven't written it yet -- what is the name -- the name of the function is going to be called Turn -- how do I function call the function turn?  Open pren turn and then provided some input.

Now I need to provide stoplight color -- the whole idea of turn is it takes the stoplight color and tells me the next color it's going to be.  The current is yellow, next it will be red.  Hopefully you can write this much.  I call my if you can with one particular input yellow and then say I expect the answer had better be red.

We're going to have some more test cases.  Go ahead and write another one to yourself on your computer at home.  Hopefully that's check expect turn of yellow light turns red and then what comes next -- I'm calling the function check—expect and giving it two inputs.

What my function actually going to reach -- I'm going to see the answer and say compare it to this and I expect a result.  Do I have some down in this?  No I don't.

Great.  Now how do I go ahead and what comes next in the design recipe.  I'll do yellow -- green, it's going to turn yellow -- what if it's flashing yellow?  What color does it eventually turn?  Probably stays flashing yellow.

Or whatever -- it's kind of a toy problem so I'm not too concerned about this.

And flashing red, flashing red is similar -- it's going to stay flashing red.  The next color is still going to be flashing red.

So yeah, between copying and pasting it doesn't take that long to type these in.  I do the first check expect and cutting and pasting will give me a bunch more of them.

Tests -- stub -- signature, header, description, what is the signature for this function Turn?

I'll split up this problem -- I'll go down the list -- (inaudible) you work on the signature -- either one type it in -- Chris and Mark try to tell me the description and Shawn, Steven and Steven give me the header of the function -- if you can remember what those are, you can talk with each other in this main chat room.  Chris and Mark the description, Shawn, Steven and Steven on header.

You might have your old notes from stuff we did last week and the homework problems we had on -- okay.  Try to -- also in general, any class before win class to the next go back and review all your notes, get up to speed on the terms.  So I'll go ahead and slam all this out but you should be able to figure this out without having to think or worry too much.

So the signature -- we decided it was the name of the function, the type that it takes in and the type that it returns.

So what type does it take in?  It takes in a symbol -- we can be more precise than that.

Yeah.  A stoplight color.

Not any symbol, it has to be a stoplight color which is one of these exact five things.  This isn't a comment up here about what a stoplight is, I'm thinking of that as a binding contract.  That is what the definition of an SLC is.  It has to be one of those five things.  If I was in Java I would have to define it within my type system.  In Racket we don't need to mention or types but we think of them anyway, so we'll have that going.

So a stoplight takes in a stoplight color -- again, my math case, F was real to real or something like that -- that's the exact same thing.  The name of the function, colon, type of the input and the type that comes back.

Okay.  Description is easy.  I'll go ahead and say in words -- finally the header is going to be say define Turn you need to give it a parameter name -- what should I call my input?  What's a good parameter name?  Yeah, I like current color.  I use the word current up here, yeah Turn of current color.  I like that.

Okay.  In fact, once I have this in here, I can go ahead and make this more technical by mentioning the parameter by name.

Now the one thing that I want to note or emphasize here, if I had a data definition that was a bunch of this or this or this or this or this, then filling in the body would be predictable.  As a stub I can type -- it better be a stoplight color -- so I'll go ahead and always return red.  I put in some answer and the point is I can press run -- some other -- this is probably what comments were removed I was talking about.

Okay.  I now get something coming up. I failed a bunch of my test cases.  Four out of five actual value red, -- it will bring me to what test case failed.  I passed my first test case.  Red is not the answer.  I like to put them there.  I figured my test cases will fail at this point.  All I have to do now is step six -- seven, complete the body expression.

We'll see in a second -- how are we going to go ahead and write this?  We're going to use COND -- I have to figure out which one of the symbols it is and give the right answer in that situation.

So this is a poster child for the use of COND.

How many branches am I going to have?  How many answer, question, pairs do I need?  Five.  Because there are five different stoplight colors and I need to handle each one of them.

There's five question answer pairs.  My first question is -- I'm going to go down the list -- I was giving something red or yellow or green -- so I'm going to say is it red -- here's the Racket for that -- I said before the only thing you can do with symbols is have some symbol I was given.  Is this equal to the symbol quote red --

So if it's red -- I'm going to fill in all the questions at this point. now they're pretty straightforward.  One question is it red -- matches this line of the data definition.  Is it yellow?  You don't need to write it yourself.  I got my pren's going on.  Green -- it will take it exactly the order so I'm not missing a case -- when I start swapping orders around, 44, 45, 46, 47, the moment I start changing the order I'm going to miss one by mistake, so.

Copy and paste can make a lot of this go pretty easy.

Okay.  I'm going to space so you can see the structure a little bit more.

Now I'm almost done, right?  Let me go back and look at COND -- if the current color is red what should the answer be?  We have our check except up here -- green.  And if it's yellow -- I'll look at my test cases it should be red, okay, and so on.  If it was green, it should be yellow.  Oops -- double clicking on a word can be kind of handy -- Dr. Racket knows what a Racket word is -- include a hyphen, so include it (inaudible) --

Okay.  So now I have this COND.  What's our last step?  Step number 8.  Profit.  What do we mean by watching test pass -- I mean test them and if they don't pass go back and fix them.

I didn't mean to have this but let's look at this -- read -- it expected a closed pren to close the open pren, possible cause line 88 -- line 88 is all the way down here preceded by a bunch of comments.  It's thinking I have a closed pren -- I didn't mean to do this but this is a perfect example -- I can see in gray what (inaudible) this all matches up and -- (inaudible) now I'll press run.  All five tests passed.

Okay.  So the upshot of that we've seen an example of using this COND construct.  It's a little bit weird.  The answers are usually more involved but we've also seen that the way we built our COND was based on the way we had our stoplight definitions.  That's one of the themes of our course.

The stub in red, what I put in red is the body so, this is -- this whole thing is sort of the header of it and this all the body of the function and I called that the stub—body and that was a transitory set.  I didn't end up with that.  I replaced it with this whole COND here.  I get points as far as you get there (inaudible) -- they actually provide a lot of the structure of the answer.  It's in a comment but all types in Rackets are in comments.  If you say I don't want that to be in comment I want that to be real code that actually gets evaluated you can look on the lecture's page and look up contracts -- yeah, I think they call them -- you can make this -- you can make this thing here -- even line 60, the turn was supposed to take a stoplight color and return a stoplight color, that's a language that doesn't have you declare your types -- all your JavaScript functions have comments about what types.  You can remove all the comments and they're just comments so it wouldn't generate an error but Barland would take off points.  Telling you what type this function handles but I better -- come on.

You might wonder what happens -- you might be wonder what happens if I try Turn of 23 -- and now I'm going to fall all the way off of COND or fuchsia or something like that -- let's find out.  Turn of 23 we got an error because I tried to say is 23 equal to the color red using the symbol equal, and symbol equal expects two symbols -- let's try turn of -- that was lower level run time error.  Turn of fuchsia -- I don't think I'm spelling that correctly -- we get a different error.  All question results are false.  Falling off the end of the COND is an error.  We're not giving back an answer at all.

By the way that's the thing when you're on the full Racket language it's allowed to do that.

Okay.  That's an example of using COND to write a function that handles a data definition that is something or something or something or something -- or means this, SLC -- I'll call it a union type.  I'll leave it up there for now.

SLC is a union type -- but what I will say is any.

(AUDIO ISSUES)

Will probably have a COND with as many branches as there are ors -- there's one last or -- up to one.  There's many branches as there are options for SLC.

Technical word people tend to use is variant.

Since now we have -- so this is coming from an idea that types are sets once we see more examples we'll take a better -- you know type is a set union -- it's a union set of several other sets.

Okay.

Yeah, so that's writing Turn.

I want to write another function.  Now I want to write (inaudible) -- for running.  So suppose I'm working at the DMV and I'm setting up something -- if they run a red light they get a fine, if they run a flashing yellow might be if you don't slow down, that's a $20 fine.  So I'm going to write a function -- fine for running.  We can do some test cases for that.

So go ahead and write one test case for this function fine for running.  Gives you what color of the light somebody ran and I want to know what they're going to owe and I'll let you play state legislature -- you can make up whatever you want for fine.  Give me a test case for fine for running.

Nobody's willing to try?  You see what test cases look for Turn go ahead and see if you can give me one test case -- make up the amount of the fine and then mark gives me one for red, Shawn for red -- someone else can give me a different one?  How many test cases are we going to have overall?  Any guesses?  I'm probably going to want to test all the different things to make sure it works so five test cases, yeah.

I'm going to go ahead and grab -- let's see how long it takes to write all five test cases -- usual like oh, my gosh so many test cases.  Start timing me now.  Copy paste is kind of nice -- fine for running.  Red light was 50 bucks.  I'll take that and one, two, three, four, five.  Red, yellow, what's the fine for running a yellow light?  20 bucks.  Fine for green light?  No fine at all.  You can go right through the green lights.  We had flashing yellow -- I'll go up to my data definition to remind myself what they are -- flashing yellow and flashing red -- flashing yellow was my $20 -- what should the fine be for run ago yellow light?  Whatever -- I'm going to play legislature -- maybe zero.

Flashing red -- that's like a stop sign -- it's not as bad as running a red light -- so maybe that's 40.

Okay.  Those are some good check expects.  They didn't take that long to do and they well define this problem.  Go ahead and give me the next steps for the design recipe -- give me the signature, the description, the header and maybe even a stub—body.

Signature, same teams as before --

(Inaudible).

(AUDIO ISSUES) --

Chase see if you can work on the --

(AUDIO ISSUES).

We did this on the previous function so you can also look there.

It doesn't quite issue a fine but it determines what the fine is.  So for description, determine the fine for running a -- color light ran.  For running a light in a particular color.

Who wants to work on the signature, Ashenafi or Chase?  Fine for running?  Takes in a symbol and an INT -- we want it to look like this -- the name of the pren, colon, you got that -- looks like it takes in two things and returns what type?  I think you were saying it takes a symbol -- it takes in a symbol and returns an INT.  You're right -- sometimes they take all the pieces of the information to return something.  We can't have something comma, something comma, something there -- here we only have (inaudible) --

(AUDIO ISSUES) -- any suggestions about INT?  Yeah, Chase it is a symbol but it's a particular type of symbol.  It has to be a SLC, a stoplight color.  I'm using SLC as a technical term because we defined what that was earlier.

Okay.  Returns an INT.  Does it have to be an integer?  I know my examples happen to be integers but can they change their mind and say next year for run ago red light next year is $50.37?  They could.  It doesn't have to be an integer.  I might say -- in Racket we'll see for reels might be numbers for integers -- it's not inexact necessarily.

What do we know about a fine?  It's going to be a real number.  Can it ever be -- yeah, it's going to be positive -- not quite positive -- 0 is not positive.  But it will never be negative.  The state is never going to pay you to run a light, no matter what color it is.  It's never going to happen.

So if I want to I can say positive real but it's not quite positive.  Not natural because it can be $50.37 -- I'll move that down here for the flashing red.  So it's not a natural number -- they're counting numbers.  It's not quite positive but it is non—negative.  If I really want to be picky.  This may be more detailed then we need but it's interesting to think about it.

Java -- in Java it just says return a double.  And you wouldn't really -- by just saying double you have a comment saying never returns a negative -- it returns a double but you couldn't completely build it in a non—negative double -- Java goes halfway helping you but not entirely, which is fine.

Now the header.  I think we had Shawn had something -- define -- fine for running but not close paren -- what do we need?  Comparing this to flying 66 --

This is now where your Java one basic programming, all right, it takes in a stoplight color and so we better get a name for that.  Pren color,, fine for the current color because I got my fine last week and it's only being processed now or something like that.

Color that was run.  I don't know.  Now I'm going to go and say I need to fill in some body here.  You're right, if I want to go ahead and do this, I should really go ahead and -- the header below -- we can do them in either order, actually -- as long as we do the tests first -- that's what I'm asking for -- you place them above or below.

So -- so we can actually run it and get an answer -- anybody have a suggestion for me to type in?  So 50 -- sure -- I don't want to think I'm writing real code -- if I'm doing that I'm going to do 09909.90 -- maybe 666.6 -- I may look at that later and say that's just a dummy thing I threw in for testing.  Now we can try running it and see if my other test cases run by fail.  Is ran ten tests, five failed, the five we finished for turn they all passed.  666 -- the expected value -- I can click on this --

(AUDIO ISSUES).

But everything runs.  If I had an unbalanced -- I can take care of it now.

Okay.  One other thing -- by the way, I want to say -- the one thing I would not really want is to return a string of my sub value -- I could but now it's really -- even if I know this isn't going to pass, at least it agree was this if it's not the correct answer it's the correct type, not non—negative real.  Once I found my parameter name, Color Ran, I like actually working that into my description.  I tend to often go back and -- this makes it really -- it's just nice -- when I read other people's documentation and the description of the function mentions the parameter by name it's really clear what's going on there.

Okay.  Now we need to go and.

(AUDIO ISSUES) --

Complete the body expression.  Handling a union type an input.  What are we going to have inside here?  A COND -- I'll be asking this -- you should think is this a union type, if it is I'm going to blurt out the answer a COND with how many ever branches depending on the type we're talking about.

In this case we can grab -- in fact, you know, I can sort of go ahead grab this whole COND from up here, paste it in, and be careful -- it's great but dangerous -- I'm going to knock out all the answers I had -- they're still the same five questions -- both functions are handling the same type of union data.  Now we can go back and fill in the dot, dot, dot -- here's my point and then we'll take a break -- I'm going to go.

(AUDIO ISSUES).

Current color for the name of the parameter?  Only because when you issue the fine -- I want the color at time light was ran rather than the current color.  If I ran the light a while ago then the current color of light doesn't matter.  It's green right now -- now it's red -- see, the light is green, judge -- that's why I don't call it current color for this function.  Think about the names and try to come up with a good name.

Back to here -- I'm going to write 29 more functions all dealing with a stoplight color -- they're all going to look like this, exempt for the name of the parameter -- I see what you're saying -- I'm sorry -- I totally misread that comment -- you're right -- my copy and paste was dangerous and I need to go and change it elsewhere.  Again, copy and now double click paste -- there we are.  Thanks.  Sorry -- that's -- I misunderstood what you were saying.

This now -- I'll fill in the dot, dot, dot -- I'll read them off my test cases in this situation.  I write 500 functions they're all going to look like this up to the name of the function, the name of the parameter.  If I take this and I'm going to go ahead and copy, paste, so another five seconds I'm going to copy, paste this and I'm going to put in block comment -- I don't even need a block comment -- here's a template for any function which takes a SLC as input.

I'll call it -- I want to remember this -- I want to copy and paste this in the future.  I'll change it to 50 and 0 and 0 and so on.

This part here I want to keep this around because next time I write a function for running -- for that handles a stoplight color it's going to look like this but I'll leave the dot, dot, dot there and I'll call it a template.  And I'll call it funk for stoplight color -- and it takes in -- it's not going to be the current color of the light -- it's not going to be the color that somebody ran -- it's going to be a stoplight color -- ASLC -- I'll remember to do this this time.

And that's where I'm going to end this part here.  This we call this the template for any function that handles a, in this case, stoplight color.  As we define different types of data it will be handy to write the template.  Suppose I ask Shawn, I want you to write a stoplight color -- write a function that takes in a stoplight color and then goes ahead -- and do it -- you didn't tell me what to do -- you can still write a bunch of the code.  The template, I can write 7 or 6 lines of code here -- it's about 90% of our characters of the previous examples.  So I've written 90% without knowing the function does.  So sometimes you're in a blank page modes, you don't know how to begin, go through and writing the test cases, writing the signature, writing stuff like that, and having the template for data type, that can be a whole lot of the actual code that's ready for you, depending on the type of the data.

Let's go ahead and take a five minute break.  Break until -- let's take it until 18:55 -- sounds good.  I'll be here to answer questions.  I'll stop the recording.

(BREAK TAKEN).

 » INSTRUCTOR:  One example I'm not so happy with -- tax purposes -- I use this for something and something and something again -- a taxable income is either a number less than 9075 or it's a number between 9075 and 3600 or a number bigger than that, which is based on the tax rate for individuals -- I can go ahead and write functions that deal with these -- I want to try something a little bit different than just that numeric type -- that will be another example of this or this or this -- I can have a COND with three branches to handle each -- each different option has a different answer -- a different way of calculating the total text.

Okay.  I'm going to skip -- I'm going to take from what we had below there -- I'm going to go and suppose I am working for the IRS -- don't hate on had he -- suppose I were working for the IRS and doing a program for running tax forms people submit.  I'm worried about line 37 or whatever the taxable income.  So I'm worried about what they enter as the taxable income.  I'm going to go ahead and call that a taxable income entry is one of something or something or -- okay.

And so I could go ahead and it could be a number they entered.  So yeah, they could have enter address number -- but maybe I have a couple of other situations.  So I'm going to say maybe it could be the symbol exempt -- maybe they're a Canadian citizen, you don't need to file taxes in the U.S. but you can say I'm a Canadian treaty and I'm exempt and that's fine.  Maybe we'll do that. And separately, there might be one more thing, maybe they don't fill in that form of their taxes.  Maybe they left it empty.  Maybe they wrote profanity that blank instead.

My program needs to take into account that any of these -- I'll stop there -- any of these three situations might happen -- it might be a real number, it might be a valid reason or invalid reason they don't provide it.

So this was step one.  I just did step one.

I had some concept, taxable income entry, I'm going to decide how am I going to reesent it.  What's going to be the symbol exempt or Boolean faults or an actual number -- positive number?  Negative number?

Any tax accountants out there?

This isn't the refund.  That's a different -- there's how much you made is income.  What is your taxable income and then how much do you need to pay or maybe get back.  But yeah, your income might be negative.  You might have lost money.  You might own your own business and have a negative income.  So yeah, or broke, it can be positive or zero.

Okay.  So but it's good to think about that -- when I say number can I constrain it to positive or negative or non—negative or something like that.

In this case it can be any of those.

And I might explain the meaning -- tax treaty, the filer doesn't need to pay any taxes.  I should probably say what false represents to real life.  Entry not filled in or filled in with a number.

Okay.  I guess one more thing about taxable income especially with numbers always mention your units, so maybe the income in dollars.

Not the income in Japanese yen or Canadian dollars.  Sometimes I might go ahead and -- this explains when I say number, Racket -- it can't and complex number but can it be a fractional number, 2.5?  Yeah.  It might be a fractional number of dollars, so I'll say real number, if I'm using Java, what type would I use here?  What Java type would I use to represent a number that might be positive or negative -- single is called float. It's because they have floating point numbers and called them float, so then they said we need double precision points and they got tired of saying double precision floating points and just called it double. They didn't call World War I until World War II started.

They just called it double.  Do we like doubles?  No.  We hate doubles.  Weird things can happen with doubles -- probably doesn't happen, most of the time it's okay, but conceivable.  If you're dealing with money how can you avoid using doubles even if you want to represents say people's income?  There's a standard CS trick.

Use number of pennies then you're down to an INT.  If you don't need to deal with fractional pennies.  Now banks when they give interest they have actual cent they need to keep track of.  But on a tax form, about the money I make is going to be rounded.  So INTs never have -- I can do INTs without over floating or errors so I like using integers when I can avoid doubles.

You may not have physical pennies -- I should say in cents -- I might have 3 cents of interest but if there's no interest I can't get paid off.

Okay. Yeah, so price of gas is whatever then they round to the nearest penny when they go and charge you.

Okay, so give me some examples -- that's step one of the design recipe -- choose a data definition. Give some samples of the data -- Steven Tanner, do you want to give me an example of a taxable income entry?  My hint is -- make sure you're in one of these three categories.

You might be away from your keyboard.  Steven Tanner, give me an example of a taxable income entry.

It's a little bit lame, I realize.  These are things I'm going to need to do a check—expect.  I'll go up to my examples of the data and grab one.

Oh, hey, Steven Tanner.

Example of the data.  One of these -- by the way, so going back over the example is when you find a stoplight color, what were examples of a stoplight color, quote red, and flashing yellow was another one -- so either Steven or Shawn, you want to give an actual example of -- one particular example.

 Either Steven or Shawn -- I'm going up the list.  So empty string -- you're saying ditto on Steven --

I'm not asking to define anything. Either Steven or Shawn -- if I say -- it's good to look at define taxable income ASLC -- that doesn't run -- it's not going to work. It looks like I'm trying to -- I'm not asking you to define a function, even if it were ASLC and parameter name, you're getting way ahead of me -- we're not talking about stoplight colors.

So oh, okay, Chris, yeah -- Chris gave the example hash faults -- is hash faults a taxable income entry?  We have a definition -- it's either quote exempt, false or a real number.  Hey, false -- that's the real category. Steven Oake, okay.

So expressions evaluate two values.

So what's another example --

And why do you know that, we said here taxable entry is one of this.  Now give a third example of a taxable entry.  Okay.  So a third example it has to be a real number.  So not quote 335, that's a symbol or something maybe.  Gosh, what is quote 335, expected a name of a symbol not that but you're right on the track so if you want to talk about 4,335, yeah, that's a real.

Okay.  So these are examples of the data.  And Shawn has some more and I'll try to talk a bit slower and louder for Steven.  I don't know if that will be enough but.

Okay, so that was step two, give some examples of the data.  And all that means reading off your definition.  You should be able to read examples off your data.  I want to read concretely about this.

Okay, and now the template.  So gosh, now, step three.  Template for a function handling type quote income employees.  I'll do this so this is going to be similar to what we did here, same idea.  Here if you look at this, this is a function that took in so here was for stoplight color so it took any stoplight color and had a five way COND, down here we're going to go ahead and taxable income entry, what should I call my parameter?  I could call it tax filled out but it could come so the places could be meaning something else, so I'm going to say any taxable income entry.  To make it clear, it's any taxable income entry so I don't know where it's coming from that's not my job so someone else is going to call it my function.  Now what goes here.  And I'll go up -- so, yeah, INT here so are we handle ago union type?  Taxable income entry is something or something or something, it's a union type, so yeah we need a COND with three conditions.  So start writing a COND with three conditions.

Okay, Chris you're all over it.

Okay.  And now what are the questions going to be?  Notice each of these is going to be a question, answer, pair and I can't give any of the answers because I don't know what this function is supposed to do yet but I know it's going to be given a taxable income entry, if someone hands me a taxable income entry I can still get some of the code, I need to find out hey, am I in the so I'm in one of three different categories here, am I in the first category, the second or third category.

How can I tell if the thing is given is in the first category so how can I tell if I was given the symbol exempt?

This is one where I hinted a little bit and if you do the multiple choice questions on the homework that may have been looked at hinted this, but it turns out we had a function called symbol, so first of all, I could do this here so are two symbols equal to each other, I can go ahead and ask, fill in this question for me here so hint looks a lot like that.  So I could say so I don't want to quite say this so why, because, this is always true.  Is exempt equal to exempt?  Yeah it's always true.  Well I'll take a cross between the two answers there.  Is taxable income entry I was given equal to exempt so if it is then good, then I found out if it's exempt or not.  So this is close except it doesn't quite work.

Why not?  We'll come so the reason is, this function a taxable income entry might be false, it might be 4,335.  If it turns out it's 4,335, let me just say symbol equal of 4335 and exempt is an error.  Symbol equal excepts to be given two symbols, not a number and a symbol, it excepts to be given two symbols.  Is there a function here to figure out if I was given the symbol exempt?  I think we talked about this last time.

Maybe this is what Chris means here so I'll see what the other people type.  Symbol is the one I was fishing for.  I'm going to ask this so I'm going to ask, hey, is the A taxable entry I was given a symbol?  Either it was a symbol, false or a real number.  If it's a symbol I know which case I'm in.  There's other way of doing this including equal question mark.

 So go ahead and let me know if I was given exempt the question would be true and fall into this answer.  If it's not true then either on false or on the number how can I tell which case I'm in?  Yeah, I can ask am I Boolean and then actually down hire I can ask so okay, so here are three different questions and now I sort of know what category I'm going to be in.  So give me any taxable entry in the world I can figure out so I know they want three different cases, why?  That's my definition.  I want three possible categories, I can use this COND to figure out which category I'm in.

Okay.  Let me sort of check Steve, are you able to follow at all?

Transcriber is not here.

Okay.

So this is step three, the template.

Okay.  This here is the template and that's step three.

Now I might write 25 different functions all dealing with taxable income entrees, this part I'll copy and paste so it's going to help.  And I'm going to end our examples so I should probably work through an example of an actual function, but I'm going to leave that here.  I'm going to go ahead and so I didn't do anything there either, so I'm going to go ahead and leave so.

(AUDIO ISSUES).

We do template steps one to three just once for data type.  That's income entry in this case.

Then steps four to seven, eight, will be repeated for each function we write, which student process something of that type.

And the reason we read the template so we can copy and paste it.  This will be the same 20 different functions if they're all handling the same type of data.

Okay.  Does that kind of make sense?  You have union data types, a something is La or blah it can be three cases, five cases, it can be different categories of things, no problem, we'll call that a union data type and we know how to start writing code for it and we notice that any function handling a data type with three branches is going to be use a COND with three branches.  It can be a union type with five branches, then they'll have a COND with five branches.

Seem reasonable?

Nobody's willing to suggest that seems reasonable?

Hearing no objection, motion passes.

We'll have these notes here.  We'll be doing examples of this, so we'll have examples of more, so I might write so I'll do a quick one.  Write a so -- so example of so write a function which takes a taxable income entry and estimates tax due zero if exempt 15% of taxable income if a real number and a $10,000 fine if they didn't enter it.

Write this function.  So go ahead and see if I can write this function in three minutes.

Okay.  We have our data type defined, what do I need to do next?  What's the next step of the design recipe?  I hear you sending it to me so test cases step four.

Check—expect.  Name this which is estimated tax due and if they go ahead and give me exempt, zero, and what if they gave me false, that's that hit with a big fine.  What if they gave me $5,000, then it's going to be 15% of $5,000 and what if they gave me zero, okay, zero.  Okay.  My test cases.

Signature.  Okay.  So signature, body, header, signature you should be following along, it should make sense.  What type does it take in?  A taxable income entry so I wish I gave it a smaller name, and returns a real number.

And return the estimated tax due.  Now how am I going to write the function?  I'm going to copy and paste this and change the name and yeah, ATIE, a taxable income entry so if it was a symbol it was exempt, if it was false it was a flat $10,000 fee, otherwise it was 15%.  Okay.

All the tests passed, including all of theses so.  So hopefully that shows you once you have the template written it's easy to go and write a function or it can be simple.  I already had a bunch of the stuff written, the characters written, I just had to fill in, turns out not many detailed to get the final answer.

Okay.  Still digesting?  It is it make a little more sense going through those steps?  So my goal is can I make up 19 more functions involving taxable income entrees and let you implement them each, and for each one should take about as much time I just spent, figuring out test cases, figuring out the signature, description, take your template, all of those 20 if you cans will go ahead and have a three way COND, all I need to figure out what is the right—hand side of each branch.

Okay.  So that's what I'll talk about there for a second.

Let's look at where are we.  Making new types by combining existing types.  Okay.  In particular I made up this new idea taxable income entry, and it was based on reals, certain symbols certain Booleans.

Your future compound types.  Another way.

Okay.  And so then we're going to go ahead and enter immediate future.  We'll have some types that are self referential, another example will be a tree.  A tree can be either an empty tree or a node with two other trees below it.  A left some tree and left some tree.  A tree is either empty or a node.  What type is that?  I use the word or in the data type that's going to be one of these union data types so it was either an empty tree or a node with information, so a union with two branches.

Okay.  Let's go ahead and take six minutes yet so I'm going to start some slides then we'll take a break, resume the slides then we'll begin compound types a little bit.

So we'll have the brain have a little rest here by sitting back and watching the beautiful slides.

So Shawn was typing so we'll have to see if this works.  My slide show goes in full screen mode.  So when I'm sharing my screen we'll have to see if that works.

Okay.  I'll mention that during the break.  And I've sent two e—mails about it now.  So you should have a look at e—mails from me.  Yes it's on the web page.

So now my question is can you all so I'm going to go into full screen mode, can you guys see a slide that says three useful categories up at the top?  And of course one of my problems I can't see your responses at all.  Let me drop out of full screen mode.  So you can all see that.  I'll do this, I won't see your answers as well, but I'll stop in about 8 minutes any way.

Okay.  Again, I'm going to step back a little bit and think about programming languages and studying different programming languages.  Earlier we saw we divided them into two different categories, functional languages and object oriented and scripted languages then we said they're kind of bogus.  Ruby is oriented and functional and procedural and scripting language, so yeah Racket same thing.

So it's good to know those terms, people talk about those.  There's some use to those things but I'll say hey, when we're looking at program language and studying them we'll look at different categories and three important things -- the first two we'll look at in this class so when you look at a language to decide if you like it or not so there's the syntax of the language so it just means the grammar rules for what is a valid program.  That's what your Java compiler does.  It says I don't care what the program does, are they remembering all the curly braces, is the word 4 followed by an open parenthesis with a semi colon in there, that sort of stuff.

So there's the syntax of language.  As a student you're like focusing on the syntax.  But it's gets simple after a while.  The semantics is important.  What is the four loop mean, what does it do, what is the meaning of a four loop.  Kind of a weird question.  What does define mean.  That sort of thing.

And then we won't talk about much in this case but a real thing you worry about using a programming language are how good are the tools that go with it, how to fully use it, libraries, is there a good editor for it, is there a good syntax for it, can you post stuff and have a good syntax color, are there other people you can share code with, are there libraries and sock etc libraries already written or do you have to do those things yourself.  Okay.  The principle, hopefully I can show you something and it does such and such and I tell you what happens and it's not surprising.  I shouldn't show you something that you think will go one way and then goes one way, like 2 billion plus 2 billion, it comes back as a negative number J Java is kind of surprising.

Okay.  But I want to spend a little bit talking about four vocabulary terms.  We'll give three and then take a break.

Value, variable and expression.  People sometimes use some of these interchangeably and they're not interchangeable.

What is a value in a programming language?  A value is sort of thing that you stick into a variable.  It's one piece of information.  So 37 or the string high, values are things that you can give to a function, values you might get back from a function, value is what you stick into a variable.  Don't confuse a value with a variable.  A variable is an identifier that holds some value.  But it's different.  Think of a box and what's inside the box those are different things don't confuse the box what's inside the box or you're in big trouble.

Values variables types.  What are types?  I can't see any of you any responses here, a type will be a set of values.  Someone was asking about union sounds like talk about sets and yeah there's a reason.  Sets of values make a type.  So Java short is a finite set, but of those particular values.

Other types like string are infinite sets.  My computer may not have memory for more than a small finite chunk of them but they're many strings but again, this is a set of values.

So we'll do all four then we'll take a break.

Value, variable type and expression.  An expression is in the middle of a program, so it's a piece of syntax if you were to evaluate it gives some particular value back.  So 3+4+5 or square root of 16 or system.in, or S is a scanner, all of those are expressions or S.next.substring 25, that's one big expression that we'll evaluate to some answer when you run it, when you evaluate it.  If you look at the word evaluate it, it has the word value stuck in the middle of it, so expression so.

So we'll go ahead and take a break there.

Let's take five minute break.  Resume slides at 7:48.  Okay.  I will see you momentarily and answer questions and I'll put up a link to that quiz.

(BREAK TAKEN).

 » INSTRUCTOR:  Okay.  Sharing my screen again.

Okay, so I guess for a little bit while Chris isn't kicked out of the library we'll cover more.

Okay.  So we talked about those -- so any questions on these or examples of these or anything that doesn't make sense, if so post that in the window and I'll see that when I pull back in a moment.

One more vocabulary terms -- I see people confuse the word valuable and variable also type gets confused -- so just be aware of that.

One other term I'll throw out here -- literalist is a value sitting in the sources code.  So going back a slide something like if you have 3+4 x 5 you won't have 16 square root until the program is running.  Square root of 16 -- a good compiler might say I can do that now and in—lining that or pre—computing that but in general a lot of things aren't computed until run time.

If you know the Mac compile time so the result of 4 x 5 -- you have the number 4 sitting in the source code and the number 5 so those are literals.  So the thing that is sitting there right in your source code.

So in Java 37 or 45 are both literals and they turn into the INT 37 -- you may not know that in Java -- it means this is a base 8 -- 045 means I have --

(AUDIO ISSUES) --

 » INSTRUCTOR:  In Java string literals they're called intern -- if I have a strange cafe in my code I'm going to call the string constructor and make that object right now in compile time you have string cafe that occurs 7 different times all in the source code it's actually smart enough to have one string object that all 7 places end up referring to.

The only reason you can do that Java strings are unbeatable.  If it's written 7 times you can use the same array of characters to represent that because I'm never going to change -- I can't change that string Java in Java -- in C you can take that string and replace it with X -- I replaced the one up here with the X but not down here.  Okay.  Maybe you can see the camera pod.  Anyway what else?  String literals of plus are at compiled time.  If you're run Thanksgiving substring of three, starting at index three -- HAY that equals return true but if you use double equal what would that do, that's not going to return true -- it will return false -- substring goes -- gets a new string and it's the string HAY with a different instance of quote HAY that I have silting in my program codes -- they both have HAY but they're two different strings -- they're dot equals but they're not equal, equals.  You remember that from our Java one.

What's interesting -- it turns out that's (inaudible).

(AUDIO ISSUES).

More over this stuff is computed compile time, but if you have two string literals and a (inaudible) -- it doesn't do that in compile time, even though the.

(AUDIO ISSUES)

Plus method -- so that's an interesting (inaudible) -- a little side note.  I'm going to stop to see if there's any comments.

Oh, yeah, I have to record.  Thank you.

Now Shawn is going to be very upset with me.  Sorry, Steve.  ITEC 380.

Okay.  Yeah, full screen mode that writes things up.  You can raise your hand to use the microphone.  I usually have it off -- when we're in full screen mode I can't see anything in the window behind it.

See you Chris.

Now, Chris will have the recording to work with -- no questions on the slides so we'll continue on the slides.  See what Shawn asks.

Statically typed language versus (inaudible) typed languages -- that's something that people often confuse with scripting languages --

Okay.  A lot of people think statically typed languages and dynamically typed languages are the same but they're all the different things. A statically typed language, what does that mean?  The types of anything that is named is known to compile.  So if you have variables called N in Java you'll know compile time.  I can look at the source code and know what it's holding an INT, double or a string.  Hopefully if it's an INT -- any way -- whereas something like JavaScript you can make a variable called N -- I can't tell you what type it's going to be holding without running it in my head.  The names can be provided by the programmer, and when you compile it actually checks that all the types match up.  There are languages where you don't write down the type but the language does go and figure out at compile time what type everything is going to have.

So (inaudible) are examples of languages that do that.  So statically type is not the same as programmer declare types but and C sharp also has something you can see bar X equal 5 and usually you have to (inaudible) but if it's real simple it will go ahead and figure it out for you.

Okay.  In dynamically typed languages, different types keep track what is a strange, what values are strings are what values are INTs or bullions, but any name is not associate with the same type.  So maybe later on it holds a string and maybe later on it holds a double.

So and P JavaScript and Racket all work like that.  What it means the value itself, the INT that you're passing around, in those languages probably has a few extra tag bits that are being kept internally, the language system keeps track of them and says this has a couple extra bits calling me it's an INT or telling me it's a string -- if you call pen something on something that wasn't a string it can still give you an error.  Just because it's dynamically type -- the types are there you just don't mention them in your program.

Duck typing, that kind of funny term.  If it looks like a duck or quacks like a duck then we'll call it a duck.  JavaScript is an example A. type of an object, your objects don't really have types.  If it has a two string method, great, we don't care what type it is.  If you call two string on an object and it didn't have two method defined then it will give you error at run time.

There are (inaudible) languages where you're responsible for making sure if you call plus on two different memory locations that those really were holding integers and one wasn't holding a string or a pointer to a memory or a character, something like that.

And finally, related term again, type safe.  A language is type safe if any type error is caught.  Okay?  So what is a type error?  When you try to do something that is illegal with that type.  So again, calling string append on asking is that isn't a string would be an example.

Java doesn't let you do that.

You might have to cash types and when you cast a type at run time it sort of -- it makes sure you want to cast object A to subject of object B -- it will make sure it can be cast to object B -- you have objects and you say I want to cast it to string it will make sure that thing really is a string.  Then you can go ahead and call string append on it and make sure it was a string.

C is not like that.  You can declare something -- you have a variable that holds an INT and assign it to a variable that holds a strange or a char star and you can cast it and the computer doesn't actually check that it lets do you that, says the programmer knows what they're doing, and then you can get errors where you're calling string append on something that was actually an INT -- bad things can happen.  It will probably keep on running too -- if you're lucky it will crash did if you're unlucky it will quit running and weird things will go down.

Those are a couple of pages of type systems and things to know about and those are separate from our scripting language.  Questions or thoughts on typing and type systems on these terms dynamically typed, statistically typed?

Ada is rich in set of types it preserves but Java is equally type safe in the technical sense of being type safe but Ada is -- you want to say like again in Java you can say something is double they can't say it's going to be a positive double, but in Ada you can -- it it's going to be integer from 500 to 509 and pass that around -- so you can take functions an integer that has to be in that range.

Okay.  Just a few more -- after these slides we'll start the next topic and then be done for the evening.  Just a couple more slides here.

This will be on the web page of course.

Strong typing and weak typing.  These terms -- people will use these in different ways, and they'll use them to talk about some of these things.  When they say strongly type they mean statically type or where the programmer has to declare everything.  Weakly where you don't have to declare it but you will get a type error when you try doing something wrong.

But we have those terms statically typed and dynamically typed.

Another way people use these is talking about implicit type conversions.  In particular in Java you can say math.square to 16 -- it looks like a type error -- it kind of looks fine but -- what is the type of 16?  It's an INT.  What type of square root might want to be given?  A double.  But when you type math.square to 16 Java has your back -- I'm going to turn this into math.square root INT to double 16 -- it rearranges the N 16 to the bit pattern -- and then it calls square root.  So it's doing the conversion for you.  And it's kind of handy.

It doesn't do a whole lot of implicit conversions.  Between INTs and doubles and numeric types and objects and strings it will convert things to strings pretty easily for you.  But besides that there's not a whole lot of implicit conversion in Java.  But PHP tries to convert everything, left, right, up, down, surveys, it hopes it will be helpful to you and if it's surprising to you well too bad.  The INT 50 and INT 16 is just fine -- when you think about in Java what happens there -- 50 plus the string 60 -- that's probably an error in Java -- if you swap it, -- put the string first it will try to put the string first -- remember I said Java does convert things to strings pretty well so I would go ahead and say 60 plus 50 and give a string 6050 -- in PHP you'll get 110 no matter what the order is.

There is (inaudible) -- if you call something to the ALU to the chip -- and you're doing a long division it will take the bit patterns and it will do long division on it.

That would be non—typed.

Now finishing up one or two more.

Compiling a program versus interpret ago program.  This is one that people tend to make a big deal out of.  It's a little bit over stated.

A compiler -- you often thinking a compiler takes the source code for a program, and produces an executable file a .A file and hopefully they're equivalent so, the compiler converts one language to another preserving the meaning of that code.

Interpreter is going to be in a function that takes an expression of your source language and gives you back what value it's supposed to be.  It's not giving you back more code to run -- it was given an expression, so yeah, 2+3 interpret language you type it, it will evaluate that for you and ask for another expression for you -- that's what Racket does.

People often get to hung up on this, make too big a deal about it.  Let's see running interpreter code -- you're running -- when the code is running you're running the interpreter which is looking at the source code as if interpret data -- running compiled code you're running on the native steal -- it's faster although it's platform specific.  Okay, true enough.  But I say distinction, but not a deeply fundamental one.  You start thinking about it more and more and here's one position -- I'll claim that CPU's are interpreters for machine code.  They take the code as data and then they go and do whatever is supposed to evaluate to -- they're supposed to evaluate content of memory location and that's what the hardware does for you.  So CPU is a very low level interpreter for machine code.

Okay.  And Java compiled interpreted, you run the Java compiler but -- let me go back a slide -- a Java compiler takes source code not to machine code, it takes source code -- it takes to Java bytecode which is machine independent, it can run on different things but it's not as fast as native machine code.  You run interpreter to run the Java bytecode you're in the JVM.  It's Java and interpreter language.

And you think of this as being a tradeoff between speed and platform independency and in general I would say that compiler is a function going back one slide again -- it takes code in one language and produces code in another language.  People talk about cross language compilers.  Hey, take some C code and give me back JavaScript code that does the same thing.  In fact, nowadays it's all the rage, people love compiling to JavaScript you take has cell or whatever you want -- Java is wildly successful great language but it runs on all platforms.  It runs on my cell phone, my computer, it runs on my car.  It runs everywhere -- so I'll write my favorite language because I can't stand writing in JavaScript but I'll take a machine from my high level language and JavaScript code -- that would be a cross language compiler.  It doesn't have to machine code it can be to every other language.

That's those slides there.  That's all I wanted to say.

Way cape first the CPR or the machine code?  Interesting.  Think about say Babbage (inaudible) -- it's not CPU but there are programs written for it.  Did he write his first programs before he had the machine written or write the -- she wrote the machine programs before the machine was developed.  They've been running algorithms for a long time.  They had the code before they ran that.

These are slides, vocabulary slides.  So there's some good general concepts to be aware of.

Okay.  We have our note taker back for Steven.  Great.

So I want to take the next again minutes or so and we'll finish.  I want to go and talk about -- I think I gave a pre—view already -- bigger picture compound types.

This is another way to take existing types and make new ones.  What was the source of a type again?  We talked about values then we said type is a set of values -- so quick question -- you don't need to do steps -- you just do (inaudible) -- we didn't need to make a new data type so we didn't need to do steps one through three.

So what is the definition of a type?  It's a set of values.  Okay, so we take the union of several sets.  We take the union of real numbers followed by the -- very small set containing just false and small set containing just exempt -- what was that data definition -- the union of three sets, one huge, the other having a essential item in them and all of those that's the set of taxable entries, that's why you call it a union type.  We take the union of several different sets.

Okay.  Let's look at compound types.

Let's switch to another window here.

Okay.  Compound types.  These are much more familiar than union types.  This is like what you did in Java after a few weeks.  I want to go ahead -- ask this in Java.  Hey, I want to represent books I'm going to write program that the library is going to use and every book is made up of compound piece of information that has four subparts, title, author a number of pages and whether or not it's copyrighted.

So if we're in Java how would I go ahead and tell Java I'm going to use a new data type called book and I want you to know that it has four pieces of information -- how would you go ahead and tell Java that?  I'm going to pull up a Java window here.

A class, exactly.  I'm going to have a class book.  Open up this.  Now what do I want about class book again.  I said the fact that -- let me see that other information -- Java I need to know every book contains a title.  Actually -- how should we represent a title in Java?  The title of the book -- there's a data type that Java has that lets me represent a title that's string.  How do I tell Java that a book (inaudible) -- I just use a field.

There are books and books contain a title that will be a string they'll contain an author which will also be a string.  Number of pages -- what should I represent the number of pages?  What are the two candidates?  INT or just in general INTs or double?  How do I decide that question by the way?  Should I represent my data by using INTs or a double?  Could I ever have two and a half pages -- can a book have two and a half pages?

And now we get into philosophical question, what would our pages -- there's a page number which is a little bit different, our natural numbers -- if I'm doing an E book I might want to say there's three and a half pages in this book.  But we'll go ahead and say -- I'll think of a physical book -- I'm old fashion -- I'm going to say INTs pages great -- is the book copyrighted -- what type in Java should I use to represents that?  Bullion.

Okay.  Is this a complete Java program?

Let me compile it and see if it compiles.

Yeah.  This program compiles.  No syntax errors.  There's no code that's running but there is information that's being conveyed.  I have a new type called book and Java knows there's a new type in the world called book and every book has four pieces of information inside of it. we'll come back to the structure in a second.  Now let me go to Racket.

We can do the aim thing in Racket.  I want to make a new type -- I'll go ahead and say define struct -- title, author, pages and is copy righted -- I get to use a question mark in that field name -- okay.  This is half of that Java program this line here is half of this Java program.  What information is in the Java program that's not in my Racket code right now?  What information is in the Java program that is not in my Racket program -- the types -- type left side be in comments but they're still important.  I'll write it like this -- just like I started in my other ones.

A book is -- I'm going to say something like this -- make book and give it a string and a string and INT -- actually I'll say and a Boolean.

And so now I have the same information in both.  I have -- first of all, this is just a comment, but at least it's in there -- there are four fields and four types up in this comment.

This is what we'll be writing in Racket.  And it corresponds exactly to this in Java.  So don't worry about it -- don't feel intimidated.  This of Java is -- we don't say class we say define struct.

Yeah, so now, what is the -- I'll continue with the Racket for a second.  Define struct actually introduces additional functions.  A constructor and four getters -- selectors maybe -- the constructor in this is named book.  And getters are named book title -- hyphen is part of the name of the getter.

By just making this is a class is there a construct for that's named for me?  It turns out just like in Racket a constructor gets names for us in Java a constructor is made for us.  In Java the constructor takes no arms and takes a empty object -- it's not helpful -- I don't want an empty object and have zero fields.  In Racket, my constructor is called make book and I'm going to go ahead make examples of the data.  This is step two of our design recipe.  This is still applying this other situation.  Let's go ahead and I'm going to make a book.  I need to provide a string for the title of the book.  I'll go ahead and provide the string -- the author of this is going to be Barland.  Number of pages -- I believe in (inaudible) that book is going to be two pages long and I'm not going to bother with the copyright.  I'm going to put it in public domain.  This is a book value -- 47 is an INT.  Quote exempt is an example.  It's also a taxable income entry.  Make book of (inaudible) is a book.  So this is one value -- and we can make some others -- another example I use is the Cat In The Hat -- maybe it's 37 pages and you better believe it's copyrighted.  Okay.

So here are two different examples of the data.  When I make test cases later on for functions involving books I'll probably copy and paste some of these to keep my life simple.

Let me go back and compare this to Java.  How way make the same book objects in Java?  Now I need a constructor that's going to help me and Java is the default constructor -- it will make a (inaudible) that's useless.  Notice the Racket if I didn't give it a constructor it makes the constructor that makes it handy -- I make book right away.

I can do this -- how do I make the constructor for book in Java?  And this again, some Java to brush up on if it's not obvious -- I'm going to make the obvious constructor that takes in for values and stuffs them in the four corresponding fields.

Here's how I'll write it.

It's copyrighted -- I thought Java was a very concise language.  This title gets -- there's more concise ways with this part here -- I'll write it like this -- I find it easy to write it consistently -- oh, my goodness -- programming is fun, right?  I don't feel like I'm having fun.

I feel I'm doing stuff the computer should be doing for me.

In fact, in some of development environments, I think in eclipse there's an item where it can do this in -- the machine can write it, if the machine can write it, it should write it.  In Java if you don't have Eclipse we have to write it ourselves.

So now can I go ahead and make examples of the data?  How do I go ahead and make the same two back books the (inaudible) and cat in the hat?  How do I make the same two objects in Java?  What is the name of the constructor?  I'm going to phrase this -- it's new followed by a space followed by the name of the class.  What is the name of the Racket constructor -- it's make by a hyphen by name of the case -- other than it's kind of the same thing.

Okay.  Something along these things -- and the only thing that I need to do now is in Java if I want this to compile I want to go ahead and put in a method, avoid test cases.  Then we compile this.  Class compiles.  It doesn't do much.  I make two objects and that's what I'm doing here.  I'll press run.  It comes back -- sorry, I goofed up a little bit -- I'm in full Racket -- it's not in student lack.  It ran this constructor and then this constructor and I got back two book objects but I'm going to change languages to beginner student.  Get rid of this line here.  Now run it.  And here's what we get back.  So here's the answer I get back is this book object.  And I'll leave you with a little bit -- what's the difference of calling the constructor and an object of that type?  I'm going to purposely confuse them a little bit or blurring them together -- I'm going to say -- I'm calling the constructor here and it creates an object internally -- when I present it I get a two string for me -- by the way in Java I can go and try to print these things -- there's no two string -- the two string that Racket provides me is kind of cute.  It's two strings that look like the call I made to the constructor.  And the nice thing about that I can copy and paste things back real easily.  And it's interesting that we can think of this as being the book object and really I'm calling a function that's going to return an object but I'll think of this as literal almost.

Okay.  Assigning something to a variable like Steven's saying is a different issue -- it's do I want to assign the result to the variable, there's that as well.  And that's cool.

Okay.  So here's what I want to you remember for next week when you're going over your notes before next week's lecture, remember -- try some examples yourself, make a different struct -- we have the defined struct and this comment line that looks -- try to represent a pet, a puppy dog and figure out what fields you want and make a couple of examples of puppy dogs.  That's your informal assignment, you don't have to turn it in or anything.  Remember what was done -- and then go and remember for the puppy dog object what is the name of the constructor and what will the names of the getters be -- calling functions and using them next week.  Okay.  Thanks Steven.  And thanks everybody.  And so you have the homework that's due.  I'll put a new homework out on this material using the COND and then making objects.  But first finish the homework from last week's material.  Sounds good.  I'll see you all next week.  And I finished on time without going too much over -- I'm one minute over or something like that.

 (Class ended at 8:34)

"This text is being provided in a rough draft format. Communication Access Realtime Translation (CART) is provided in order to facilitate communication accessibility and may not be a totally verbatim record of the class."