Transcript (Intro to Debugging)

I would like to talk about debugging now this is a topic that I'm an expert in because when I write my code I make lots of mistakes and I end up spending a lot of my time fixing those mistakes when I first was learning how to write code nobody ever taught me how to do this I just kind of figured it out and and found some things that worked for me you're probably in the same boat you've probably been making mistakes in your code since about the first line of code you ever wrote and whether you realize it or not you've come up with some things that you do uh to help you fix those mistakes my goal here is to give you some concrete strategies some things that maybe you hadn't thought of uh that will help make you a little more efficient at finding those bugs and getting your programs back on

track when we say bugs we what we really mean is just an error or a flaw in your program so we'll start with a quick bit of history and and trivia that I always think is interesting um the term bug predates this story that we're we're talking about here but um when we started using it in the context of computer programming was in 1947 and uh Admiral Dr Grace Hopper who was a Computing Pioneer um was having trouble with a computer that wasn't behaving the way she uh was expecting she opened it up and she found a moth that was inside there um kind of disrupting the electronics so Hopper and her colleagues um took that moth out they taped it into their little log book and they made a note saying that this was the first actual case of a bug being found and ever since then that term kind of stuck and we talk about these as being computer

bugs there really are two types of bugs or two ways we can describe bugs in our programs uh there are compile time errors and there are runtime errors and they each present kind of different challenges and different different ways that we have to debug them so the first type of error is a compile time error and this is when your compiler can't translate your source code into machine code that the that the computer can actually run they're caused by syntax errors or kind of semantics so for example if you're trying to write an if statement uh in Java you need to have those parentheses if you forget the parentheses you've violated a syntax rule of the language and the compiler stops it doesn't know how to handle that uh in Python an analogy would be maybe leaving off the colon at the end of an if statement uh because that's a that's a syntax rule in Python and similarly the python compiler or really interpreter um won't know what to do with that and it won't won't let the program run often syntax errors are just really simple mistakes like again forgetting semicolon forgetting to put a semicolon at the end of a Java statement or not indenting properly in Python in any language if you misspell the name of a variable or a method or something like that that's a syntax errors error for the most part syntax errors are identified in your IDE you've been seeing those little squiggly lines um similar to like misspellings in Microsoft Word or something like that even though your IDE can identify these a lot of times they can be really tricky to fix for couple of reasons um one the error message that you see may not make a lot of sense especially if you're new to programming it's you know it's sometimes there sort of techno Babble right um another problem the second problem that makes these hard to to fix sometimes is that uh sometimes the compiler message doesn't even point to the correct line right you might make a mistake on line 10 but the compiler is able to keep going after that and it's not until you know a few lines later that it finally gets confused and realized it doesn't know what's going on and so it it thinks the error is down here but really you've made a mistake uh way back on line 10 um and then sometimes you know one error May hide another error and so you fix one and you realize there were actually you know five other errors that it wasn't even getting to because the compiler kept stopping um so those can be kind of tricky they have to be fixed before your program will even run uh so they are the in they are sequentially the first kinds of errors that we encounter the other type of error we have is called a runtime error or we also call these logic errors and these happen when our program doesn't execute the way that we want it to uh runtime errors can have different symptoms different ways that we know there's a problem for example it might crash or give us an exception and and cause the program to actually stop running uh but often the program does run it just maybe gives us incorrect results or something in the program doesn't behave the way that we expect it to behave you have undoubtedly experienced runtime errors in apps or games that you've used um I use the example in a game of if you have an NPC that's like glitching and getting stuck in a wall or something like that that's a logic error that's a runtime error uh it's not stopping the game from working maybe but it's not behaving the way that the game designers and the programmers intended it to work now runtime errors can be really really tricky to fix your IDE can't detect a runtime error because the code you've written follows all the syntax rules and it will execute or at least it'll start to execute uh so as far as the compiler is concerned the code is fine and there's no error as an example maybe you are uh going to add two numbers together but instead of typing a plus sign you accidentally type A minus sign well the IDE sees that and that's perfectly valid the compiler sees that no problem I can do that the computer doesn't know what your intent is it just follows whatever instructions you give it you know the program is getting the wrong results so you have to go and find where your mistake is without the help of the compiler or the IDE now I hate to say this but if if your program doesn't work correctly it really is your fault right we've all had that experience where we know in our hearts that our code is correct you've been looking at it for 20 minutes and you can see that it's right but it's still just not working the way you want it to work you have to remember a computer is just a device that does exactly what it is told to do if you're getting the wrong result it's because the source code you created is flawed that can be really frustrating and discouraging but you have to remind yourself there's always an explanation for your program's behavior and unfortunately it's probably not because there's a dead moth in your computer shorting things out um the good news is if we understand that the computer is just doing what we told it to do that means we can fix it we have we can find that error and we can correct the mistake because debugging can be so frustrating and so confus using uh my goal here is to give you some tips and some strategies to help you um figure out what's going on and get those errors fixed and get back to coding as quickly as you can this is going to start with really using a coding process or development process that makes you less prone to make errors in the first place and makes it easier to find and fix those errors when you do so this is going to start as soon as you really begin that first line of code in your ID e and you have to accept that making mistakes and having to fix them is just part of the process in fact it might be the most important part of the coding process every programmer makes mistakes from beginners to the most accomplished programmers in the world if you've had me in an inperson class and you've seen me writing code live you've seen me make all kinds of mistakes and then I fumble around trying to figure out what's wrong you're probably sitting in your desk looking at the projector going I can see right there what the mistake is okay for me good code doesn't just pop out of my head fully formed good code is the result of a development process and that that process includes making some mistakes so stop beating yourself up about it and just accept that this is what we

do the next thing you can do as far as having a debugging mindset and a mindset that minimizes the amount of error errors you make is to run your code often run it every time you make any kind of a significant change or you make significant progress even though you're absolutely certain that nothing you did is going to be a problem you run it anyway uh if you type up 100 lines of code before you try running it the first time and then you get an error you now have a 100 lines of code that you have to go through and find that mistake but if you just type a few lines of code and then run that if you do have an error you only have a few lines of code that you have to look at if something goes wrong so run your code frequently don't let yourself build up a bunch of errors that you have to fix at once just work on a little piece at a time and on that same note when you start on a programming project whether it's something you're doing yourself or an assignment for a class think about how you can break that up into small tasks right and the idea is you want to work on just a little tiny piece and you get that piece working before you move on so maybe you start out with I'm just going to get the users's input I'm going to prompt them I'm going to get the input I'm going to save that in a variable and I'm just going to get that working before I move on to the next thing um it's really part of that same idea of of running frequently and focusing on little uh little bits of code at a time uh another thing you can do to help break up your task into um smaller more manageable pieces is to use methods or functions um kind of liberally right use them often um identify the little tasks that you need to do in in order to solve the big problem and then for each one of those tasks write a method or a function that just does that a lot of times you can write and test a function by itself and you can get that function working completely and then you can incorporate it into to the rest of your program once you know it's

working okay so you've got the right attitude you're you're working in small pieces and fixing errors um right away and not letting them build up um they're still going to happen right so what do we do once we have an error and we need to deal with it so how can I approach that the first thing I recommend that you do is clean up your formatting right when you're working on a section of code and you're making lots of changes to it or or making lots of progress a lot of times our indenting and our use of white space can get a little bit messed up in a language like python this can directly cause errors since indenting is how we specify blocks of code but even in a language like C messy indenting and poor use of Whit space can make it hard to visually identify how your code is running so in this python example um this second print statement is supposed to be part of what executes if that if statement U comes back as true but I haven't indented it correctly so it's not going to work properly uh in my Java or C example here of an if statement because I've put these curly braces in an odd spot and I haven't indented properly it's easy to miss that well that that block ends right there and it leaves out that that second print line statement or it would be a right line statement in C uh and so it's really easy to miss that most idees have a feature that will clean up your formatting so just use that um a lot of idees including Visual Studio code have a setting where you can um make it automatically format your file every time you hit save I would encourage you to turn that on um the other thing to point out here is aside from helping you debug clean formatting is really really important for just general readability and you need to make sure that you clean up your code before you submit your work and and by that I mean submit it to your teacher or if you're in a workplace U before you kind of submit that pull request to have that code um be merged into the project you need to have it cleaned up

first uh my next tip is to stop and work the immediate error uh don't gloss over an error with the intention of oh I'll just come back later and get that one this can compound your problem right you can uh make other errors because of that error and they start to build up so just stop and focus on that one error until you get it fixed and don't move on sometimes a fix requires a pretty big rewrite and any additional work that you've done before you fix it might end up being useless anyway so it just ends up wasting time similarly if you find yourself in a situation where you have a whole bunch of Errors you need to just pick one and focus on fixing that one error instead of trying to fix it all at once um I usually start with whatever I think is the most important error but you could also just you know focus on whichever error comes up first in your program it doesn't matter but just pick one error and work that error until you got it solved the single best tip I can give you the most important single piece that you can take away from this is to give yourself a way to see what's happening in your program when your program is running you can't see what the variables are are doing you can't see how program execution is Flowing so you don't know for sure where things are going wrong in a different video I'm going to show you how to use some of the tools in your IDE to help keep an eye on those things but there's a really easy way to do this without getting into fancy tools and that's just to use a ton of output statements right um and by that I mean and if you're in Python print statements if you're in Java then print line or or right line statements uh for C so for example prior to a big calculation just print out the value of all the variables are in that are involved that way you can see are the mistakes happening before the calculation or after the calculation in a loop print out your counter or your accumulator so that you can see how it's iterating um print a message at the top of a method or a function I can't tell you how many times I've written a really good method and then forgotten to actually call it and I put a print line statement in there and I run my program and realize I'm never even going into the

method these examples here show an infinite Loop right so I run my program and it just looks like it's frozen nothing is happening well something's going on in this Loop so if I come back and add a little print statement or print line or right line statement then I'm going to see that that number is just the same every single time I have forgotten to increment my uh my counter for my Loop so that's an example of how using output can just show us and give us insight into what's going on in our program and I like to say that output statements are free they don't cost anything so use them often you can always go back later and delete them when you fix the issue or comment them out if you think you might need to be debugging something uh similar in the same piece of code

later if you're working on a big program it's smart to try and isolate that problem the um try and run just the code that is giving you trouble so we don't obviously want to delete things that we've written and that are part of our program but you can comment out a lot of stuff that maybe isn't relevant to where you're seeing the error so for example maybe you skip the part where you're getting user input from uh from the user and you're just assigning a value that for one thing that saves you from having to type stuff in all the time um when you run your program and you can focus on just the code that that has the problem if you're calling a bunch of unrelated methods and by unrelated methods I mean things that aren't related to the problematic code just comment those calls out um so they they get skipped your program probably isn't going to work the way you want it to work but the piece of code that you're trying to debug you can run that and focus on just that

code nobody wants to hear this one but my next advice or my next tip for you is to document your code right very few programmers like to write code comments or documentation but if you're looking at this code and you really can't figure out what's going on take a few minutes out of your life to just write a single code comment for each line of code explaining what it is is supposed to do that forces you to look at each code and analyze each code and don't just write what you think it's going to do look at it and write what it's really doing um something about that just shifts your brain enough to where you can kind of see oh man this line of code that I've been looking at all this time has an error that I just didn't even notice um this also has other advantages that are going to help with kind of The Next Step I'm going to tell you and and it can help down the road if you have problems with this code again later at least you've got it all documented out and you can understand what you're supposed to be doing uh and try and figure out why it's not working the way you

want the last tip I have for you is the easiest one but it's the one that people always forget and that is to just take a break right you stare at a problem and you just become more and more and more frustrated um something happens when your brain uh is like that in that mode where you you just kind of stop recognizing your code you're seeing what you wanted to write instead of what you actually wrote It's the strangest thing but I see it all the time um you're just seeing what you want to see uh on some level I always think maybe you're you're sort of emotionally invested in the code like you've spent a lot of time on it um you don't really want to accept it's wrong you don't want to delete that code and start over so your brain is just telling you that it's okay when it's clearly not um um I don't think it's productive to sit there for 20 minutes looking at the same lines of code over and over so do something that's going to give your brain a little bit of break from debugging I don't care if it's you know taking a walk or wasting a little bit of time on Reddit or working on a different program go to sleep and start again tomorrow just do something that's going to clear your head shift your attention onto something else even if it's just for a few minutes you will be amazed at how you can come back uh from grabbing a bik to eat and you sit down and instantly you see what the problem was it was there all along and you just were missing it um I also like to point out that this is a kind of a life proo tip not just a debugging tip um I always tell students when they're working on their resume write your resume get it absolutely perfect and then set it aside don't look at it for a week when you come back a week later and look at it you're going to see all kinds of little mistakes and typos that you hadn't noticed before it's just something about how our brains work right um one of the best ways to get fresh eyes on your code is to have someone else take a look at it right that other person isn't frustrated by the code um they don't have any investment in in whether the code is good or not or how much time you've put into it they're just looking at a piece of code in class this happens all the time a student sits there staring at their code for a long time um stuck I can see they're stuck I walk over and I take a glance at their monitor and I can tell right away oh you you typed in the wrong variable name that doesn't happen because I'm any smarter than you I'm not any smarter than you it happens because I've got fresh eyes and it also happens honestly because I've made that mistake 10,000 times before so I recognize it you're you're only making it for like the first or second time so you don't you don't realize it yet if you do find yourself stuck on a bug for more than 10 minutes or so I think that's probably also a good time to just reach out for some help that could just be that other person that you call over to take a look it also might mean um raising your hand in class or emailing your instructor scheduling time with a tutor um and unfortunately for that that means you can't be a procrastinator right if if you're doing your assignment 20 minutes before it's due it's hard to give yourself that break or to reach out from uh for some help from somebody but as much as you can if you can give yourself just a little bit of uh something else to think about for a few minutes I guarantee that's going to help you find the error a little bit quicker now just as kind of a final thought to leave you with I would really encourage you to just be patient and to cut yourself some slack when you're debugging remember that something is causing the problem and it's just a matter of finding it and keep in mind that um even great programmers make mistakes and they get frustrated when they're debug debugging so this doesn't mean that you're bad at this or that you're not you know you're not a good program or something like that debugging is a skill just like anything else and you're going to get better at it um try using some of these suggestions when you run into problems and maybe you'll start to find some strategies that really help you um be less frustrated and more efficient with your coding