Section 1

ENGLISH ORIGINAL TEXT УКРАЇНСЬКИЙ ПЕРЕКЛАД

|  |  |
| --- | --- |
| JASON HIRSCHHORN: Welcome. So just a few announcements before we start. So sections, everyone should have sectioned already. Sections normally run from Sunday to Tuesday. So you guys should get your assignments. And then I think next week you'll be with your actual TFs. So if you have any questions, or trouble, or you forgot to section. Some people do that. Just let us know. Shoot us an email at heads.  Then office hours. We started office hours last week. Office hours every week. Monday is in Leverett 8:00 to 11:00. Tuesdays in the quad, so Cabot 8:00 to 11:00. Wednesday, Mather 8:30 to 11:30. And Thursday Annenberg 8:00 to 11:00.  So Scratch, a lot of people don't need office hours help for Scratch, which is completely fine. If you do, that's absolutely fine as well. But in the future, the problem sets, they get much harder. So this is going to be your friend. Definitely go to office hours. Plan on going to office hours. When you get to problem set four, five, six, seven, you need to go to office hours. That's where a lot the help and the work gets done.  I think last year, just for example, on Thursday night-- so I think the problem sets were due on Friday. So Thursday night, we would have 200, 250 students at office hours. So definitely make use of these. These are your best friends. Like this is where if you're stuck on a problem set, this is where you'll probably get help. So office hours. OK so those are announcements. Announcements are done, so let's start.  OK, the appliance. Did everybody download the appliance yet? No. As in so so. I kind of downloaded the appliance. Right so the appliance is new this year, so we're probably going to have some bugs in it. So please download it as early as possible. So problem set one is going to tell you exactly how to download it. Getting started installing right here. It's going to tell you how to download the appliance.  So definitely download the appliance sooner rather than later. Instructions are in the P set spec. So if you wait until Wednesday night, and then you have trouble, and you send us an email at midnight on Wednesday night that you can't download the appliance, that's not a legit reason for an extension. You need to do it now, and you will have trouble. I tried to download it last night. I ran into a little bit of trouble. If you do run into trouble installing, definitely hit retry, because I just hit retry a couple times, and eventually it worked, which doesn't make any sense, but it does. So definitely retry, but if you come up against a brick wall, shoot us an email at heads, and we'll be more than happy to help you guys. What's up?  STUDENT: Just a quick question. If they check style, there's a part that says we didn't include this. JASON HIRSCHHORN: Not yet. STUDENT: It says execute it. How do you execute that to make sure it's in your system?  JASON HIRSCHHORN: You're going to run a bash command. I'll go over that later. Cool. So the appliance can be a little bit intimidating, because you're used to kind of running in graphical user interface. So Scratch was a graphical user interface, so what do I mean by that? What I mean is that when you're programming, you're basically using blocks of code. You can see the code and stuff like that.  The appliance, you're going to be doing things in a command line environment. And so for the rest of your programming lives, you're going to be doing things in the command line environment. So it's good to jump in early and get some experience. So let's do it. So that's Rob.  OK, so let's just jump into command line stuff. So this is the appliance. Can anybody tell me why we use an appliance? What is the appliance? Does anybody know? What's up?  STUDENT: It's just a virtual machine that you run your machine so that you can get over the OS disagreements between Mac and PC.  JASON HIRSCHHORN: Perfect.  STUDENT: It's a universal template.  JASON HIRSCHHORN: Yeah, it's like a universal template. That's perfect. So yeah, it's a virtual machine. So it's basically an operating system. This is an operating system. This is equivalent to your Mac, or you PC, or your Linux. This is the exact same. And so you can do similar stuff. If you wanted to, you could go online for example. You can't really see it, but right here, Google Chrome. You can go online if you want to. This is an operating system. And the reason we do that is because it's much easier when we're handing out instructions and everything. If it's just a uniform environment, it's much easier for us, and it's much easier for you. You're not going to run into it run into any idiosyncrasies when you're programming. We know exactly what you're going to run into. So the appliance is here.  So when you're starting programming, you're going to go down to the left hand corner. There's a little box. You're going to click it. This is your terminal window. So this is where you're going to be a lot of the semester. So let me zoom in a little bit.  OK, so getting around a terminal window is a little bit different. First, there's no icons. I can't click anything. There's nothing to click. So you have to figure out a way to figure out where you are, what's here, and how to move around.  So the two most useful commands probably are ls-- so what do you think ls does?  STUDENT: List. JASON HIRSCHHORN: Lists, yeah. It just lists algorithms within the directory. And then cd. So let's say I wanted to cd CS50, what did that do? Yeah?  STUDENT: Changes your directory. JASON HIRSCHHORN: Perfect, yeah. It just changes the directory. So let's go back. So how do I go back? So let's say I wanted to go back to the previous directory. Yeah sure.  STUDENT: I think you write cd just without anything. JASON HIRSCHHORN: You can do that. So perfect. So if I write cd, this is actually going to pop me back up to the home directory. So you see this tilde, that's going to pop me way back up to where the tilde is. So it's going to pop me up back to the home directory. But let's say I did something like this. Let's say cd CS50. ls again. I've got other stuff. cd super section. cd file, so I'm going deep. And then let's say I want to just pop back up to the top. cd. Let's say I don't want to do that. Let's say I just want to pop back up to the directory that was right above me. How do I do that? So let's say cd CS50. cd supersection. So let's I'm here. Let's say I just want to pop up one time. How do I do that? Yeah, what's up?  STUDENT: You just type cd supersection again.  JASON HIRSCHHORN: You could do that. Well I'm in supersection.  STUDENT: Oh, the one before.  JASON HIRSCHHORN: Yeah, so you could do that. You do cd ~CS50. So if you look here, this is your path right here. That's your path. So you could seriously just go cd ~/CS50. Boom, you're there. What's an easier way to do it though? That was perfectly correct, but what's a slightly easier way to do it, because a lot of times you're going to be jumping in directories, jumping out of directories. So let's go back. cd supersection. So let's say I want to get back real quick. You can do something like cd dot dot. That's going to pop you back up real quick. So cd supersection. cd files. Let's say I'm here. By the way, clear is good, because my window was getting a little bit messy. So ls, so I don't want to be here anymore. cd dot dot. Take me one up. Cd dot dot takes me one up. Cd dot dot and back to my home.  OK, so cd, ls. Probably the most important things, because you're going to need to know where you are. ls is going to tell you where you are, and cd is how you're going to jump around.  You can also do some other cool stuff. So for example ls. ls is just going to show you the directories in your file. It's also going to show you the files. But it's just going to show you what's there.  If you wanted something a little bit more cool, and you're going to actually do this in a couple weeks. ls dash l. That prints out a little more information. Can anybody guess on the left hand side the drwx, rwx and stuff like that. Does anybody have an idea what that might mean?  STUDENT: Permissions.  JASON HIRSCHHORN: Yeah, perfect. Permissions. So you're going to have to deal with this when you do web programming later on. If you've ever gone online, and you've clicked on an image or clicked on a field, and it said permission-- like permission not allowed, or like permission not granted or whatever, that's because when they're programming, they haven't set these permissions to let you do that. So where that comes from.  OK, so that's cool. So we jumped around. We can look at stuff in our directory. But how do we make stuff? I've got directories here. That's awesome. How do I make a new directory? Say I want to make a new directory for right now. You can do something like this, mkdir. So make directory. So what do you want to call this directory? STUDENT: Awesome. JASON HIRSCHHORN: Awesome. I love it. Make directory awesome. And there you go. So we need a list. You have a new directory. And you can tell it's a directory in your appliance because it's got this color, right? So it's like a bluish purple or whatever. So let's change. Let's change directory. Let's go into directory-- yeah, what's up? STUDENT: Is there a list of all these [INAUDIBLE] online? JASON HIRSCHHORN: Yeah, I'll put them up on-- we'll have slides. It'll be a PDF. We'll put them up for everybody. Yeah, sorry it's a lot of commands. if you want me to slow down, or go back, or anything like that, definitely let me know. It's a lot to absorb right away. What's up? STUDENT: Can I ask a potentially stupid question? JASON HIRSCHHORN: There's no stupid questions. STUDENT: There are. A directory is for just looking at information where you store it.  JASON HIRSCHHORN: Yeah.  STUDENT: Is that essentially what you're doing? So when we make something, hello world or whatever, it's stored in the directory, yes?  JASON HIRSCHHORN: Yeah.  STUDENT: Is that true for everything. Everything is stored in the directory.  JASON HIRSCHHORN: Yeah. Whenever you make it, whatever directory you're in, it's going to basically store it in that directory. STUDENT: And when we're calling it, we're calling it from the directory.  JASON HIRSCHHORN: Yeah, so the directory is where it es. So it's kind of like if you have a new folder on your desktop and then you write a text file, put it in that folder, that file is in that folder. Does that make sense?  STUDENT: Yeah.  JASON HIRSCHHORN: Yeah, what's up?  STUDENT: Can I just ask you to make it a little bit bigger?  JASON HIRSCHHORN: Oh, yeah. Sorry. The projector is so bad. We had so much trouble with this earlier. Is that better? Is that too big?  STUDENT: No, no it's not.  JASON HIRSCHHORN: No, it's not. Is it too small? What's up?  STUDENT: I'm sorry, what's the command for clearing?  JASON HIRSCHHORN: Oh, clear. If you want to clear, just type clear. clear enter. On the last appliance, what it basically does, it doesn't clear your information, it just kind of shift everything down one screen. So if you typed clear-- for example let's say I'm scrolling up. I can scroll up. This is everything I did. But if I type clear, and then I scroll up, here's my stuff. So it's not erasing it, it's just basically giving you a new frame.  STUDENT: How do you [INAUDIBLE] a directory?  JASON HIRSCHHORN: We'll get there.  [LAUGHTER] Slow your roll. All right. OK, so so we're in a directory right now. So David, I think in lecture he did some cool stuff. So let's write a program. So you can do that a number of different ways. You can use any kind of text editor. You can use Nano, you can use Vim, Emacs. Those are slightly more complicated. Or you can use something in here that's called gedit. Sorry. gedit is not happy. So gedit is basically just a text editor. Just like I text edit word processor or something like that. So it's got a nice interface just like this. So you do something like that. So actually let me give it name because I never did that.  So what do you want to name our first program? It's just going to say hi.  STUDENT: [INAUDIBLE].  JASON HIRSCHHORN: You lack imagination. OK, gedit. Hi. I'm going to do .txt. OK, so here it is. So let's write a program. So I think you saw this in lecture. You need a main function. Boop. So this is just a main function. Bigger? Yeah, sorry. OK so you need a main function. I think I'm jumping the gun a little bit, but it's never bad to get used to this kind of stuff.  So a main function, every C program you're going to write for the rest of this course will have a main function. Does anybody know why? What's up?  STUDENT: Start.  JASON HIRSCHHORN: Exactly, so it tells your program where to start. So if you don't have a main function, and you compile this and stuff, it's not going to know were to start. Main is always where it starts. So you have a main function, and then does anybody remember how to print?  STUDENT: printf. JASON HIRSCHHORN: Yeah, printf. printf hi. OK, now I have set you guys up to fail, but will this compile.  STUDENT: No.  JASON HIRSCHHORN: Why? There's multiple reasons why. Sure.  STUDENT: It doesn't have the standard [INAUDIBLE].  JASON HIRSCHHORN: Perfect. So what's this called up here?  STUDENT: [INAUDIBLE]  JASON HIRSCHHORN: It's a header file, right? It's a header file. And you're exactly right. Both of you are right. So to include that, I just do pound include standard input output .h. Am I good to go now? Would this compile?  STUDENT: No. JASON HIRSCHHORN: Why not? STUDENT: It's a [INAUDIBLE] text. JASON HIRSCHHORN: Yeah, awesome. So I just made it. So I saved it, but gedit hi.txt. So now if I do ls, here it is. Right here. Oh, it's not in awesome. I must have gone out. Oh, perfect. How do we put this in awesome? So I don't want it here in my home directory with everything else. I want to put it in the folder. How do I do that?  Not too hard. Move. So we've already learned ls, list. cd to change directories. cd dot dot to bump out of directories. And now we're going to start moving around files. So to move just move hi.txt, and I'm going to put it in awesome.  Just be careful. Whenever you're writing in the command line, everything matters. So uppercase matters. If I had done lowercase a, it's not going to know what's going on. So everything matters to be very specific.  So run that ls. It's no longer here. Let's change directory into awesome. ls. Boom, it's there. And somebody mentioned why this would not compile. Why is not going to compile again? Yeah?  STUDENT: It's not in a .c.  JASON HIRSCHHORN: Perfect. It's not a .c, s so how am I going to rename it to a .c? What's an easy way to do it? Let's use the command we just did. So we're just going to move hi.txt to hi.c. ls hi.c. And then just for fun, let's just jump into this real quick. hi.c. You'll notice it's nice and colorful. So these are just kind of helpful colors when you're programming. So the int and the void, those are data types. We'll go over those in a second. Hi is a string. But useful colors. So if you ever accidentally choose the wrong extension, you should have these colors. What's up?  STUDENT: How can I copy into a file [INAUDIBLE] into another file? JASON HIRSCHHORN: Oh, I'm going to show you that too. So copy remove are on the dock. Let's make this. Make sure it works. Make hi. Works, run it. So ./hi awesome. That is your first program. Let me make it bigger. So a really good-- Yeah, what's up? STUDENT: Why would it be that when I take gedit, it doesn't give me another line for gedit hi.txt? It just puts the cursor to a line. STUDENT: Yeah. The [INAUDIBLE].  JASON HIRSCHHORN: Wait, so say that again.  STUDENT: When I type gedit, the first time, the first line-- JASON HIRSCHHORN: Up there at the very top? STUDENT: Has no J Harvard appliance, and the cursor is just all the way to the left. JASON HIRSCHHORN: All the way to the left? STUDENT: Yeah. Does anyone else have that?  JASON HIRSCHHORN: Oh, well J Harvard is just like the username. So that's just the standard username.  STUDENT: It stops responding. JASON HIRSCHHORN: Oh, it stops responding. OK, so what you have to do is you have to close out of gedit. So close out of gedit, and it should return. So is gedit open on your appliance right now? If you exit out of it, it should return and you should be good to go. Did that work?  STUDENT: So did you use the gedit command to open-- JASON HIRSCHHORN: No, the first gedit command was a mistake. The first gedit command, I to do gedit hi.txt. So I give it a name, so gedit hi.txt. hi.txt is the name of the file. The first one was just a mistake, and then I quickly closed out of the program. So the question was if I do gedit, I'm not going to be able to really do anything else in my command line until I close that gedit, so I think that's what they ran into. So close the gedit. It'll return out of that program, and then you'll be able to actually type in the command line again. Is that good? Dope. So just make sure to close it if you run into trouble.  OK, so a really good question a second ago was copying. So now I have hi.c. But let's say I want to change it. I want it to be slightly better. I want to say hi class. But I want to keep this template. I don't have to rewrite the entire program again. I want to keep the template. How do I copy?  Easy. So cp hi.c newhi.c. Now I have two. And then I can open up newhi.c, and instead of just saying hi, I'll say hi class! So I'll go back. Exit out. Go back to my command line. ls make newhi. Run it. There. It's better, much better. So if you want to copy something, easy as that. cp is copy.  So we've gone over moving, making directories, copying. Let's see what else should I go over? Oh, yeah. Let's go over this. So this was awesome. Let's say I wanted to-- oh wait. I'm going to go back into awesome.  Let's say I don't want these files right here. Let's say I want them in a new directory like in directory awesome. I want to start organizing, so now I'm going to have a new directory called hi. It's got all the programs I write called hi. So how do we do that?  STUDENT: Change directory.  JASON HIRSCHHORN: Yeah. hi. Boop. mkdir hi\_dir. It complained because I already had a hi in here. So now I have a hi directory. So now I want to move everything, all the programs I wrote, I want to move them into the hi directory. Just clean stuff up. How do I do that?  STUDENT: [INAUDIBLE].  JASON HIRSCHHORN: Moving it, right? So let's move. Yeah?  STUDENT: What's the difference between hi and hi.c that we already have. JASON HIRSCHHORN: Oh, hi.c is the actual. So that's the file that you wrote. So if you were going to open it up in gedit or Nano, that's what you're typing in. When you compile it, then you get an executable file. That's hi. So if you opened up hi, that's going to be a bunch of gibberish to you. It's basically going to be bunch of computer instructions. What's up?  STUDENT: What do all the colors mean? The blue is probably directory. What's yellow?  JASON HIRSCHHORN: Those are the executable files. So when I did make hi.c, it gave me-- so I can do it. Jumping the gun. So remove hi yes. So then let's do ls. I don't have hi anymore. So when you make, that's when you take the program that you wrote and you make a executable file. So make hi.c-- well, make hi. I think he went over this in lecture.  These are bunch of compiler instructions. So instead of having to actually write clang with all these. These are flags. You just write make. So this makes the executable files. So when you do ls, now you have hi. So this is what you're going to actually run. And so you run that by doing ./hi, and it runs your program. But if I try to do that with hi.c, no, it's not going to work. Yeah, does that make sense?  STUDENT: Mm-hm.  JASON HIRSCHHORN: Cool. Wait, I wanted to say something else. Just real quick, in case anyone is interested, when you jump out of a directory, it's the dot dot. Does anybody have an idea of what maybe just the singular dot means? STUDENT: [INAUDIBLE]. JASON HIRSCHHORN: It's actually the current directory. So it's your current directory. So I'm here. I'm in awesome. If I do cd dot, it doesn't do anything. It takes me to my current directory. So a little redundant, but you have to do that, so ./ your program name, in order to run your program, you have to do that. OK.  STUDENT: If you do cd dot dot dot, however many dots will it just take you up that many directories?  JASON HIRSCHHORN: You mean like cd dot dot dot dot? STUDENT: Yeah. JASON HIRSCHHORN: No. So actually Remember I did ls -l? If I do something else, I can ls -a. So what this says is -a shows you everything. So it shows you things that are hidden and things that aren't hidden. So if you notice that when I did just ls, I just saw that stuff, the stuff I made. If I do -a, it's going to show me everything. And so current directory, directory above it. Does that make sense? Yeah?  STUDENT: But those aren't actually in the directory. The current directory isn't in itself, so why show those with dots inside?  JASON HIRSCHHORN: You mean it's like it's not in the directory.  STUDENT: Right. So we have a list of things that are in the directory.  JASON HIRSCHHORN: I think it is in the directory. Like -a shows everything, even if it's hidden inside the directory. So I think it's just a route to the current directory if that makes sense. It's a little recursive. It's like Inception. What's up?  STUDENT: So you said -a shows [INAUDIBLE]? JASON HIRSCHHORN: -a, yeah.  STUDENT: What makes it [INAUDIBLE]? JASON HIRSCHHORN: Well, for example, let's go back up to the top. It's usually stuff you shouldn't mess with and you don't need to worry about. So this is kind of the home. If I do -a, I get tons of stuff. All that stuff. And you don't need to mess with any of that stuff. Definitely don't start going in here and just remove stuff.  [LAUGHTER]  Yeah, this was a little bit more comfy stuff, but it's good to know. I think it's good to know. But if you never want to type -a in the entirety of this course, just forget it. What's up?  STUDENT: What is the ls -l? JASON HIRSCHHORN: That shows you permissions. Basically you use that for permissions, so -l. It shows you a little bit more information, but again, this is a little bit more comfy stuff. These are just permissions. So the permissions of the folders. So just for example, let's say I made a folder, so the d represents directory. So those are directories. And then read, write, execute. So those are permissions for I think user, group, world. Totally going too far. You're going to go in this like weeks from now, but just so you know, those are permissions. What's up? STUDENT: Can you clear rather than just skip to the next thing, can you-- JASON HIRSCHHORN: You mean actually delete? I don't know. I never do that. Why would you want to do that? Why would you do that? I like it. You live on the edge. You're like I don't want any of this. That's intense. STUDENT: [INAUDIBLE]. JASON HIRSCHHORN: Yeah, we can do that. Sorry, messing with the size is totally freaking me out right now. Let's go up. Yeah, so now I can't go up anymore. So everything I taught you is gone forever. Thank you. We made our directories. We made our files and stuff like that, so we still have all that stuff. OK, I'll go fast.  So real quick. So let's say I'm in awesome. Let's say I wanted to move everything to the new directory because I just want to organize it, right? So how would I do that. Move. We're going to use move again. So move hi.c. And what you can do is you start listing stuff, and the last thing you list is where it's going to move it. So move hi.c, hi newhi newhi.c, hi\_dir. And so it moved everything into the last thing you mentioned. So then change directory, hi\_dir ls. Everything is in there. So it's nice and more organized.  OK, let's say I hate my original hi program. I want to get rid of it. How do I get rid of it? What's up?  STUDENT: [INAUDIBLE]. JASON HIRSCHHORN: Let's say I just want to remove the program, so just hi. No, you're right. So it's rm, but you do rm hi. It's going to give you a little warning. It's going to say are you sure you want to delete this, and yes I do. If you don't want to be prompted like that because you don't have time for warnings, remove -f. This is another flag. It's like the -l. It's like the -a. These are just flags that you're giving to these commands. So remove -f. -f f means force. So force it. I do not want to be prompted. So remove -f. Let's do hi.c. Just got rid of it. It didn't tell me.  But let's say instead I'm done with this directory. I want to get rid of directory. So I do remove hi\_dir. No, that doesn't work. No, so it won't let me remove a directory. Does anybody have an idea?  STUDENT: Is it remove dir? JASON HIRSCHHORN: So remove like this?  STUDENT: No. Just the [INAUDIBLE]. JASON HIRSCHHORN: No, right now there's no hi. There's just hi\_dir. So this is the directory. So actually what you want to do is-- oh, do you have an idea? I see like a half hand.  STUDENT: [INAUDIBLE]. JASON HIRSCHHORN: OK.  STUDENT: [INAUDIBLE].  JASON HIRSCHHORN: OK. So if I want to remove a directory, remove -r. That's recursive. So remove that directory recursive. So I want you to go into that directory, remove everything in the directory, and then get rid of the directory. And then I'm going to f as well, because I don't want it to prompt me with warnings after it removes every single file. So remove -rf hi\_dir. Boom, it's gone. Just be careful where you use that. I'm going to use it here too. -rf awesome. It's gone. Be careful where you use that. If you use that here and type something like this, don't ever do that. That will delete everything on your appliance without ever warning you. And I've done that. Grading problem sets at 4:00 AM, I was trying remove problem set directories, and I did this. And I was like oh my god. Don't do-- I tried to cancel it. You can cancel. Command C just cancels everything, but I already deleted like 75%. So I ruined it. And people do that. Everyone does this. Don't do this though this semester. It sucks. I'm nervous having it here. I'm going to delete it.  [LAUGHTER] Can you imagine? OK, questions. Yes?  STUDENT: So if we do everything you're doing here, is this the same as just going into actually file save as [INAUDIBLE]. JASON HIRSCHHORN: Oh, you mean through gedit. STUDENT: Yeah. JASON HIRSCHHORN: Yeah it is. gedit is useful. So gedit is very useful. In this class, you can do a lot of things through gedit. You can save. You can rename. You can move files and stuff like that, but just truthfully when you go to 50-- well you are in 50-- 51, if you go into 61, this is going to be your life. This is the command line. This is where you will program for the rest of your life, so definitely use gedit if you want. Definitely a good way to start. For the first problem sets, definitely use, but every once in a while try to get more used more comfortable with command line arguments. But you can, definitely. What's up?  STUDENT: So when we want to delete one of the files we've made, there are two ways. [INAUDIBLE]. Is there a way when you're deleting a directory to ask for prompt so you don't end up deleting it? JASON HIRSCHHORN: rm -r. So don't force it. So rm -r. It's going to go through recursively, delete everything, prompt you after everything, which will drive you nuts. Like I almost exclusively do rm -f and rm -rf, just constantly. I'm like I don't have time. I'm a busy man. All right, what's up? Yeah.  STUDENT: Is there an undo, or a trash bin, or anything like that? JASON HIRSCHHORN: No. [LAUGHTER] I could go back and get my appliance that I just deleted, no. No, no, no. If you do rm, it should be gone. What's up?  STUDENT: So you can [INAUDIBLE] together. JASON HIRSCHHORN: Yeah, just like that. So for example, you saw the rm -rf. Let's say I want to do ls -la, it shows me everything, even hidden things with their permissions.  STUDENT: So then r, the -r flag, if we don't have that, we can't delete a directory, right?  JASON HIRSCHHORN: Right.  STUDENT: So that says that we want to do something with the directory? Is that what that means?  JASON HIRSCHHORN: -r means recursively. So a directory is going to have files in it, right? So what you want to do is you want to go into that directory, delete everything, all the files in that directory, then pop out and delete the actual directory. So the -r means recursive, do it all recursively if that makes sense. Yeah.  STUDENT: So directories have [INAUDIBLE] and just delete it. Or do you still--  JASON HIRSCHHORN: Let's try. So mkdir empty. Remove empty. No.  STUDENT: No, so you still need the--  JASON HIRSCHHORN: Yeah, always. So -r, -rf, empty. Awesome. Anymore questions? I think that is your crash course on command line stuff. Any questions, because we're going to jump over to data types, and loops, and all that stuff? What's up?  STUDENT: I don't have CS50 showing as one of my directories. Is that normal?  JASON HIRSCHHORN: Oh, yeah. That's mine. Yeah, that's where my solutions to your stuff is.  [LAUGHTER]  So yeah, that's absolutely normal. I think you'll probably just have these four. I think that might be normal. Yeah. OK, any more questions? I know that was super fast. What's up?  STUDENT: I saw the blue green color. What is that?  JASON HIRSCHHORN: That means directory. STUDENT: And this blue?  JASON HIRSCHHORN: Oh, you mean this blue green?  STUDENT: No.  JASON HIRSCHHORN: This blue green?  STUDENT: No. When you listed-- JASON HIRSCHHORN: Oh. Like this?  STUDENT: Yes, the icons.  JASON HIRSCHHORN: Oh, this. I don't know. What does that mean? So it seems to be pointing to a directory.  STUDENT: It almost looks like an alias.  JASON HIRSCHHORN: It might be. You don't need to know that. Anymore questions? Awesome. If you have any questions, I know that was a lot. A lot of this information will be online. I'll put it online for you guys. Section B will be online as well. Or if you have questions, just come up and ask me. Here you go. So now Sharon and Hannah will go over some of the logic behind problem set one.  SHARON: Oh my god. Troubles.  HANNAH: In the meantime, I'm Hanna. This is Sharon. We are both CS50 TFs, and we're going to give you a little intro on a couple important things for problem set one. Everything from data types to conditionals to for loops. Loops in general.  SHARON: And in the end, we will take a look at P set one, and what you have to know for it. So let's start with data types. You guys should be familiar with the ones that are highlighted in blue. So we can start with ints. ints are integers, so one, two, three, four. floats, floating points. Like integers with decimals, so 5.2 or even 5.0. Chars are characters like A, B, C. And a string, you guys all know. Like CS50 or hello and world.  Bool is a Boolean, so we have true and false as our Boolean. So if something computes to truth-- 5 equals 5, that computes to true, and if we have 5 equals 4, that's false, so that computes to false. And here we have the associated size of all of these data types. And you will have to know this for your quizzes, so this might be helpful to remember, but we'll post these. Actually they're already posted. So you should be familiar with the ones highlighted in blue.  JASON HIRSCHHORN: Real quick, just in case you want to know the difference between a char and a string. So when you're writing stuff, whenever you have something like "hi," that's going to be a string. So if there are two quotes, it's a string. But if I were to an "h" like that, that's a string. But if I only have one quote, that's a char. And so what's the difference you might ask. Well, the difference is this is one byte of memory. This is 4 bytes of memory.  SHARON: float? Describe it again? OK, it's kind of like a number that has a decimal point with it, so 5.2 or even 5.0 is a float versus just 5 is an int. STUDENT: [INAUDIBLE].  SHARON: Oh, it can have many decimal points, so 5.675 to 3.1415. Yes?  STUDENT: Do we need to know all the sizes for this? SHARON: Yes. STUDENT: [INAUDIBLE].  SHARON: You don't worry about the white ones just yet.  HANNAH: The notes are posted at CS50.net/sections. STUDENT: Just out of curiosity, is there a utility to knowing the size of the files you're calling so that we should know it?  SHARON: So these are the data types, but yes. So in this case, which one would take up more memory or more space?  JASON HIRSCHHORN: So for a concrete example, in I think problem set four or five, we're going to give you a bunch of data, and you basically have to traverse that data. So it's useful to know that if you have so many integers, that's going to be what. So let's say I gave you ten integers, what's the size of that? STUDENT: 40. JASON HIRSCHHORN: 40 bytes, right. So you might need to know that OK, I need to jump 40 bytes. So let's say instead it's 40 chars, then-- let's say I gave you 10 chars, then you know, OK, I only need to jump 10 bytes. So it's very useful to know the size of the actual data types, because a lot times you'll be jumping around the data, so you need to know how far to jump.  STUDENT: What's the difference between the quotes and one quote?  JASON HIRSCHHORN: OK, two quotes is a string. One quote is a char.  STUDENT: Like operationally, what's the difference? They both say hi. JASON HIRSCHHORN: Oh, sorry. This is just h. Sorry, this is not clear.  STUDENT: So the top one-- JASON HIRSCHHORN: This is a string, yeah. So it's two characters, right? String, hi. But let's say I just had a single character, if I put two quotes around just h, that's a string. This is a string h. Does that make sense? So this is four bytes. But let's say I got rid of that an only put one. One quote. This is now a char, a char h. So exact same thing, different data type. This is only one byte, whereas the string was four bytes. Yeah. STUDENT: So if you were trying to store a word like David, you would want to use a string because regardless of how long the string is, it will only take four bytes unless it's a long long or a double, in which case-- so if it extends beyond four characters, then you want to make it into string. Is that the general logic behind. JASON HIRSCHHORN: If it extends beyond one character, it's a string.  STUDENT: So there's no [INAUDIBLE] to storing a four letter word as four individual or three letter word as three individual characters because that will only be three bytes, as opposed to the three part string. JASON HIRSCHHORN: You could do that. You could do that, but I don't think that's really worthwhile, because you're only saving one byte. You know what I'm saying? In the big scheme of things, that won't really matter. But for example, printf, if you're printing f and you have three chars you can print that out-- let's say you C-A-T, you can print out cat just by doing one char one char one char, or can you print out string cat. It's the exact same thing. So you could do it like that, but--  STUDENT: It's not going to save you that much.  JASON HIRSCHHORN: Yeah. The headache that it would induce is not worth it. Yeah?  STUDENT: For example, you had the h with one versus two, the single or double quotes. Why would you want to save a single letter as a string. What purpose does that ever serve? JASON HIRSCHHORN: I don't know. I mean, it really depends on the program. For example, I think later on you'll get into command line stuff. So for example, instead of me asking you for an integer-- sorry, can I speak into you? HANNAH: Yes.  JASON HIRSCHHORN: So for example, instead of me asking for something with get end, which I think you might have seen a lecture, there might be a command line. You just type it in at the command line. It would be like -/hello4 or something, or helloh h, whatever. You do it at the command line instead of being asked for it, that command line is always a string, whether it's a four or an h, that is always a string. So that's one example of when you might have a singular letter or a singular number represented as a string. SHARON: All right. And then here are some basic operators. Hopefully you're familiar with the first four just in general. So there's adding, subtracting, multiplying, and dividing, and make sure you use the right keystrokes.  And then there's also modulo, which some of you might not be very familiar with. And what modulo does is that if we take this example 4 % 2, it takes the remainder of what that does in division. So 4 divided by 2 is 2, and there's no remainder. 4 divide by 3 is 1 and 1/3, and so remainder 1. So it computes to one. And then 4 % 5 is the fraction 4/5, and the remainder is 4. Does that make sense? OK, cool. And we follow PEMDAS here too.  So some Boolean expressions. So you guys have seen equals equals to compare, say, two numbers. So if 5 equals equals 5, that computes to true. Make sure you have only one equals sign. For not equals, it's bang or exclamation.  HANNAH: Two equal signs when you do equality.  SHARON: Oh, yeah. Make sure you don't have one equals sign. And then for not equals, it's bang equals. And then you can look through for less than, greater than. And then we have logical and and logical or right here. And what that does is, if you remember in Scratch when you had that block that said and, and then you could fit two different pieces there, that's what this and does. And make sure you do two ampersands. And this key you can find towards the right, right under your Delete button. Question?  STUDENT: What was the [INAUDIBLE]?  SHARON: OK, so if you only do one, you're comparing bits. So if you remember what bits are, they're 0's and 1's, and so that's comparing something else. So we're going to focus on this for now.  JASON HIRSCHHORN: So one equals is like assignment, right? So like int x equals 4. You're saying that variable x equals 4. So 1 equals is assignment, but the ands and the pipes or the bars, just like Sharon said, they're bitwise operators. You don't need to worry about that.  STUDENT: When do you use logical and and logical or again? I just forgot when you do that.  HANNAH: Sure, so if you ever want to check if two things are true, for example, let's say I want to check that a number is between 5 and 15, I would say make sure that the number is greater than 5. So let's say if x is greater than 5 and less than 15. So in order for that whole statement to evaluate to true, you need both of the kind of sub statements to evaluate the true. With or you only need one of the two or both. STUDENT: Great, thank you.  SHARON: And also in Scratch, I'm sure in a lot of your projects you had if touching the edge or if touching another sprite then bounce or something. HANNAH: All right, so why do we care about all of these Booleans? We have these structures that you've seen in Scratch called conditionals. And conditionals are anything of the form if some condition or some Boolean is true, then do the code between these curly braces. So you can see on the right here is the Scratch block. You have this if then, and anything that goes into that little shape that looks like this-- i don't know what you'd call that-- that's going to be the Boolean or condition.  So again, a Boolean or condition is anything that either evaluate to true or false. And again, you can combine Booleans. You can have, again, x is greater than 5 and x is less than 15. Or you can just have one of those x is less than 5.  OK, so in C It's on the left. It's just the keyword if. In parentheses, the condition or the Boolean. And then the code in between those two curly braces will only execute, will only run if that condition or that Boolean is true. Does that make sense? Awesome. OK. And then as you may have seen in Scratch, we can also add on an else, which is basically anything between the curly braces under else will only execute if the condition is false. Make sense? Any questions on these two? Awesome.  Cool. So here's a quick example. Let's say we want to determine, based on the time, whether we should say good morning or good evening. I'll say if it's before 12:00 noon, we're going to say good morning, otherwise we're going to say good evening. And when I said that otherwise, that's equivalent in C to this else. So we're going to check if the military time is less than 12:00, say good morning. Otherwise say good evening.  STUDENT: So we'd actually get that input for military time with the get string. HANNAH: Right. Did Somewhere else we'd actually have to provide that information. Right here it has no value. We never even declared it. I assume that somewhere above this program I declared the time, or asked the time, or-- anything else with this example? Awesome.  OK, now we have in addition to that simple if else format, we have two other different structures. So first we have the switch statement, and here's the general form of a switch statement. You have the keyword switch the same way you have the keyword if. And then an input, in this case n. So this can only work with integers. So that input n has to be an integer, OK? And we're going to follow one of these cases depending on what that value n is.  So in this case, you first compare is n equal to constant one, if it is, do everything kind of indented over here. I can point to it because I'm too short. If it's not equal to constant one and is instead equal to constant two, we're going to follow that second block, and we can do this for as many times as we want, and then that default is if it didn't match any of the above cases, execute that code. Any questions here? This one is a little more complicated. Yeah?  STUDENT: Break signifying what?  HANNAH: Sure, so once we enter that block, if we find, let's say, our n is in fact equal to constant one, we enter that block after the colon. We do whatever is on that line of commenting, and then we break, meaning we get out of this switch statement entirely. STUDENT: OK.  HANNAH: Mm-hm?  STUDENT: Is the indentation necessary, or do you need [INAUDIBLE]? HANNAH: Sure. So, as always, indentation is not strictly necessary. The computer doesn't care whether you indent. We, as people who are reading your code, do care. So it's a lot easier to look up at the screen and see, oh, I know exactly what happens when n is equal to constant one. If I had random indents, I wouldn't be able to tell so easily. Yeah?  STUDENT: So if wanted a user to choose one of the options that we have on the list, would we use, I guess, [INAUDIBLE], but get an integer from them and put that into n? HANNAH: Exactly right. So let's say we wanted to check what integer. We said oh, enter an integer between 0 and 5, let's say. We could ask for that n, ask for that value, and then have each of those cases. And again, this is something we could very easily do with if statements, right? We could have if equal to case one, if equal to case two, if equal to case three, so on and so forth. This is a little bit faster and a little cleaner. It's kind of just a nice structure now. STUDENT: Faster as in writing? Or does it run the computer a little faster?  HANNAH: It's runs a little faster. Yeah?  STUDENT: So the double dash is just comment or--  HANNAH: Oh, I'm so sorry. I didn't point that out. OK, if you have that //, is a comment. Just // is a comment. So any time the computer sees that, it's going to say OK, anything that follows this, I'm just going to ignore. I'm going to pretend you never even wrote that.  STUDENT: So this [INAUDIBLE].  HANNAH: This does absolutely nothing. But if I wrote something like on the first comment I wrote instead printf say congratulations you entered constant one, that would be something. Yeah?  STUDENT: So in real life, the case would just simply be a number and then [INAUDIBLE] would be another integer.  HANNAH: Exactly.  STUDENT: And do you need the ellipses?  HANNAH: Sorry?  STUDENT: Do you need that?  HANNAH: Oh, no. I'm sorry. That was just to indicate that you can go on for as many cases as you want. Let's do a concrete example that might make things a little more clear. OK, so let's say I say, OK, give me an integer n that represents a class number, specifically a computer science class number. So if you give me 50, I'm going to say great. CS50 is an introduction to computer science, and then I'm going to break. So that means I jump out of this whole switch statement, so now I'm done running the code, OK?  If you gave me instead 51, I'd print the second statement. And then if you gave me some number that wasn't 50 or 51, I'm going to say sorry, I'm not familiar with that class. Yeah?  STUDENT: You don't have break.  HANNAH: I'm sorry, I don't have break?  STUDENT: What if.  HANNAH: Oh, what if you don't have break? Excellent question. So what would happen is you would go in, and you would check am I equal to 50? And let's say, yes, you were equal to 50, you'd print the statement. And then you would continue executing, so you would say am I equal to 51? And you would go on and go through every case like that? Yes?  STUDENT: Is the default line analogous to saying else? HANNAH: Exactly. Very good. It's like the kind of catch all.  STUDENT: So if you didn't have break and then one of the case statements was true, and then it said increase n by 1, then it would automatically make the next one. It would check case 51, and then display that as well? HANNAH: Yeah, I think that would work. So you could kind of get messy, so a break is a good thing to have. Yeah? STUDENT: Without break would it do the default? HANNAH: That is a good question. JASON HIRSCHHORN: I think it just runs. So if you don't have break-- so let's say I did 50 and it checked 50, and it would print that out. Print out CS50 as introduction to computer science. You don't have break, it should go and keep going until it hits a break. So if there's no break, it's going to keep going. It's going to print everything else out.  HANNAH: So I guess that would be including the default.  JASON HIRSCHHORN: Yeah.  HANNAH: Good question. Yeah? STUDENT: Is case considered a function? HANNAH: I'm sorry? STUDENT: Is case considered a function? HANNAH: Is case considered a function. I would hesitate to say that.  JASON HIRSCHHORN: So normally when you see functions they'll be in curly braces. So for example, when you look at your code, for example, main. It was main open paren, then void, close paren. Functions-- they're basically always you'll see parentheses. So case if case doesn't have any parentheses, that's your clue that that's not a function. But it's not. Yeah?  STUDENT: So then does that make switch a function?  HANNAH: Does that makes switch a function.  STUDENT: In the sense that you put something in, it gives you one of the cases. JASON HIRSCHHORN: I wouldn't call it a function. Not everything with parens is a function. I mean that's a clue that it is a function. I wouldn't say switch is a function because it's not really returning anything. You'll get into that more.  HANNAH: Yeah, just think of it as an if structure. STUDENT: Can you nest this in an if structure? HANNAH: Yeah, you nest pretty much anything.  STUDENT: Barry said the quotes are sort of like the same as else. So can you replace the quotes with else?  HANNAH: Not in this particular structure because the switch statement is expecting the word default. The computer knows that default means something special, that means the catch all. Anything else on this. We have one more to get through. Yeah?  STUDENT: [INAUDIBLE]. HANNAH: Difference between using a semicolon and a colon. So a semicolon is always to tell the computer I am done with this line. You can go ahead and execute it. This is a complete line. A colon is going to, in this case, bring you into a particular block. So semicolons are always use to end lines. Colons are used for a variety of other cases.  STUDENT: Is this equivalent to just saying if n equals 50, then blah blah blah. HANNAH: Mm-hm. STUDENT: It's just the same [INAUDIBLE]. HANNAH: Exactly. And we said it runs a little bit faster. If we did not have the break, it would be like if, if, if. If we do have the break, it's like if else if else if else if. Cool?  JASON HIRSCHHORN: So when coding, you can just use if else, but I think last year on the quiz we had them write a switch statement, so just in case. HANNAH: But definitely nothing wrong. We're not expecting you to worry about performance crazy things. Just if else is totally fine. These are just good things to be aware of.  And here's our last one that kind of falls under this conditional category. We have something of the general form condition ?, and then a little piece of code that happens if the condition is true. And then colon, a little piece of code that executes if the condition is false, semicolon. We're done with that line. So syntactically it's a little bit ugly. We're going to go through an example I think that's most clear. So we want to assign the string professor to one of these two values, either David Malan or not David Malan, OK? So you should be familiar with the string, professor =. We're going to assign a string to the variable called professor.  Now we want to check a particular condition. In this case, our condition is class\_num = = 50. And now might be a good time to point out when we have string professor =, that's one equals sign. That's assignment. Whereas in class\_num = = 50, that's two equal signs. That's a quality check. So we're going to say is the class number equal to 50? If so, assign Professor David Malan. If not, assign professor to not David Malan. Any questions there? Again, this is just something that's good to know. You could do this with if else. A good practice problem might be to, when you go home, right the same exact conditional in an if else form, because you can do that. Any questions here? All right, I think we're going to go on to loops. Awesome.  SHARON: OK, let's talk about while loops. So first on the left you see here-- OK that says while, while (condition). And then you do this-- can we change this?  JASON HIRSCHHORN: It's cut off.  SHARON: OK, and then there are also curly braces. So pretend that curly braces are there above and below the do this again and again. So if the condition in those parentheses evaluates to true, then you should keep doing whatever is in the while loop.  So for example, we could do a practice problem. I'm wearing two earrings right now, so let's say if I'm wearing at least one hearing, clap your hands. So if we're going to go through this, we're going--  [CLAPPING]  Right? And I'm wearing one, and-- OK. Yeah. I'm not wearing earrings anymore, so no more. OK, so then that's when you would stop. And this you could say is equivalent to almost variables. Let's say you have a variable number of earrings. So while number of earrings is greater than or equal to 1, clap hands. And then after clap hands, decrement earrings. Do like earrings minus 1. So decrement the number of earrings, and then you will go through the while loop twice.  And if the condition is always true-- so if, let's say, 2 = = 2, and 2 is always equal to 2, right? Then you would always do something in there, and that's equivalent to almost the forever loop that we had in Scratch. Yes?  STUDENT: Are the two sides just different ways of writing a while loop?  SHARON: So we're focused on this one first. And then so let's compare it with a do while loop. So they're slightly different. OK, so let's say the condition is still while I'm wearing at least one earring, and I'm not wearing any earrings right now. And let's say do and still clap while I'm wearing at least one earring. What should happen?  [CLAPS]  Uh oh, guys. OK, so you're supposed to clap once because basically you go through the first part of the code. You do it no matter what, and then you see the while condition, and you go back into that loop if it's true. Does that makes sense?  STUDENT: You always do it the first time. SHARON: You always do it the first time regardless of whether or not that condition is true or not. JASON HIRSCHHORN: When do you think you could use this? When does it make sense to use this? HANNAH: Yeah? STUDENT: [INAUDIBLE]. JASON HIRSCHHORN: Right. HANNAH: Very good. JASON HIRSCHHORN: So when you prompt a user, you're going to always want to prompt a user one time. You're always going to want to prompt the user one time. So instead of putting in a while loop, you put in do while loop, because you're always going to do it one time. If they give you the correct answer, you're done. If they don't, then you re-prompt them.  SHARON: All right, for loops. So in Scratch, we had repeat blocks. So we wanted to repeat something, let's say, seven times. So we just said repeat seven, and say I'm here to help you Snow White! In C, we have for loops if we want to go through something a specific number of times. Let's say if we initialize the variable dwarves and make sure-- so the first block right there before the first semicolon, we initialize our variable to, and we set it to 0. And our variable there is an integer, int. And the variable name is dwarves, and we set dwarves to 0.  And the second part between the two semicolons is our condition. And so as long as dwarves is less than seven, we'll keep going through this for loop. And then the last part is what do we do at the end of this for loop? dwarves ++, and that means we increment dwarves by one every time.  So what's going to happen here? So first we're going to go through. We have dwarves as 0, and then we're going to print I'm here to help you Snow White! And then dwarves is going to increase because we said dwarves ++. Dwarves is going to be 1. And then we compare-- dwarves is 1. We compare it to is dwarves less than 7? Yes. I'm going to go through this again. I'm here to help you Snow White! And then dwarves becomes 2, and then we compare it. Is 2 less that 7? Yes. We're going to keep going through. And we're going to go through this seven times.  So in the end, we're going to have dwarves = 0 print out I'm here to help you Snow White! dwarves equals 1, 2, 3, 4, 5, and 6. Remember we index at 0. So we start with 0. Yes?  STUDENT: So this is different than the do because this doesn't print out initially at first. So you could continue the same thing. Could you do this also with the do loop? Like the same process. The condition is dwarves less than 7 or whatever. Response is less than 7. SHARON: So you could technically. So if we go back to-- your talking about the while loop, right? So the do while loop is slightly different because we guarantee at least once that we'll go through it, so that's the biggest difference. But with the while loop we could say while dwarves is less than 7, do this, and then increment dwarves by one. And then we have to initialize dwarves before this whole statement that it equals 0. So yeah, we could do that with that. Mm-hm?  STUDENT: Can you go back to [INAUDIBLE]? SHARON: For loop? STUDENT: Yeah, so with the dwarves ++, seems like that's what you do after you've ran some other means. SHARON: Correct. STUDENT: Could you just not have that and put it-- SHARON: You could. STUDENT: So is [INAUDIBLE] right after print out in the next line underneath. SHARON: Yes, you could put it there. But then you would just leave that empty.  JASON HIRSCHHORN: You still need the semi colon though.  SHARON: It looks a little awkward, but you could technically do that. Technically. Please don't. Yes?  STUDENT: Are there any alternatives to ++? Is there anything else [INAUDIBLE]?  SHARON: Technically it it's dwarves. One equals sign we're going to set it to dwarves plus 1. So technically that's what dwarves ++ means. Does that make sense?  STUDENT: Yeah, but are there any alternatives? Like if you ever--  SHARON: Yeah, you could do dwarves - -.  STUDENT: [INAUDIBLE].  SHARON: Yeah, you could do a host of things. HANNAH: And you could increment by 2. You could increment by 3. Anything that's going to be changing, it will eventually make the condition.  STUDENT: So if you wanted to increment by 2, how would you write that?  HANNAH: You could either write dwarves-- you could write this whole thing out. dwarves = dwarves + 2. Or a slightly shorthand. I'm just going to write + equals 2.  STUDENT: OK, and you'd write that right where dwarves ++ is there. HANNAH: Exactly. SHARON: Correct. There was another question? Yeah?  STUDENT: [INAUDIBLE] dwarves ++? SHARON: So you don't-- JASON HIRSCHHORN: Yeah, don't need it there. [INAUDIBLE] very end. So you do [INAUDIBLE] ones because you're kind of splitting up the initialization, condition, and the change. At the very end, you don't need it. SHARON: Also notice you don't need the semicolon after the whole entire for loop.  STUDENT: How would you start with an initial number of negative, for example?  SHARON: You can initialize dwarves equal to negative 2.  STUDENT: Do you just do dash 2?  SHARON: Yes, so the negative sign, dash 2.  STUDENT: Is it [INAUDIBLE] to initialize as dwarves [INAUDIBLE]?  SHARON: Yes. So if we had just initialized dwarves earlier, just do int dwarves semicolon, and then there we can do dwarves set that equal to 0.  STUDENT: Could we do it earlier in the program saying int dwarves = 0, and then just--  SHARON: Just not have-- so there would be a space again, but you would still need a semicolon. Yeah. Mm-hm? STUDENT: Does this code say what the value of dwarves is afterward. If you ask it after all of this is done to print the value of dwarves. SHARON: You could ask it to do that. STUDENT: OK. SHARON: But you would have to ask it to do that. It doesn't do it for you. I'm scared to put my earrings back in. Do it at the end for the applause. Just kidding.  JASON HIRSCHHORN: So that's the same. So initialize at the top. Condition, print, change. So this is exactly the same as that if anybody can see that. STUDENT: Why don't you have the semicolon on the first line? SHARON: Where? STUDENT: After the first line. SHARON: After the 4 in the parentheses? Because we want to go through into this loop. And if you take a look at loops in general, they don't have semicolons. It's not practical. Are we good? One more question, two more. Yeah? STUDENT: This might be a little bit advanced, but do dwarves have a different [INAUDIBLE] from a variable [INAUDIBLE]. SHARON: Very good. STUDENT: [INAUDIBLE].  SHARON: Yes, it is. What? JASON HIRSCHHORN: Yeah, so it does. So normally the scope of something is in the curly braces. So scope is always going to be in the curly braces, but that wouldn't make much sense for the for loop because we initialized dwarves in that curly brace. So normally the scope of dwarves, that variable wouldn't extend past that. This is a special case though. So you initialize it within those parentheses, and then you have the scope later on. So special case. They should not have that, and that was-- That make sense?  STUDENT: No.  JASON HIRSCHHORN: I can go over it. So what are you still confused about?  STUDENT: As in can you use dwarves-- you can obviously use dwarves [INAUDIBLE] outside the formula too?  JASON HIRSCHHORN: No, no. So the scope of it is only inside the curly braces for the for loop, yeah. SHARON: But if you initialize dwarves outside of the for loop, then you can use it elsewhere There's one more question? No? OK.  So we talked a little bit about nesting for loops or nesting conditionals, if statements. So here's an example of where we could nest a for loop. Let's say we were printing this table of x's right here. We might want to-- first if we just don't look at the code and we just think about it, we want to go through every row and print each column, right? Does that make sense? OK.  So here we're going through every row for basically each row, and there are three rows. And then within each row for each column, and there are four columns, print an x. so. When row is 0 and column is 0, we print this x. And then we keep going through the column loop. Row is still 0, but column is one. And then column is 2, and then column is 3. And then we exit out of that loop because then column is no longer less than 4. And then we print a new line, and we go to the new line. And then we go through the next row, and row gets incremented, and we go through that again. Does that make sense? Yes?  STUDENT: So to nest [INAUDIBLE] is just putting it inside of the for loop?  SHARON: So nesting means that we have a for loop within a for loop, like Inception.  STUDENT: You don't need a special rotation or anything? You just stick it right inside of it? SHARON: Correct. Yes?  STUDENT: Might be [INAUDIBLE], but it looks like there's an extra space between all the x's. I don't know if that's-- If that were to actually do the program, would it do something like that?  SHARON: So no. That was pressing Enter. That was bad on our part. Apologize.  HANNAH: How would you change this program if you did one extra line?  STUDENT: [INAUDIBLE]. SHARON: Good job. STUDENT: You can also print the same thing by switching around the column, right?  SHARON: Hm?  STUDENT: So to print your little xxx, the first line can say print, and instead of saying row, it says column.  SHARON: So you could change the variable names to do that. Is that what you're saying?  STUDENT: No, I'm just saying that-- so you're printing rows first and then printing columns. Can you also print columns first and then rows to get the same diagram? SHARON: You could, but then you would have to be careful of where you put the new line. And how would you jump back? JASON HIRSCHHORN: If you just switched it and the values were different, instead of having four-- what is it? Three by four. I You would have four rows and three columns. Does that make sense?  STUDENT: Yeah.  JASON HIRSCHHORN: So yeah, you could do that. Yeah, definitely. SHARON: But that would be literally just switching the names of the variables. Good? OK. All right, P set one. OK, so the first part of it you will be doing this, Mario. Just kidding. It's more like this. And so when we just looked at the nested for loops, making that table, think about how you might be able to print out these hashtags in this way. And then right here how would you print this right here, this open space.  STUDENT: [INAUDIBLE]. SHARON: Yeah, you just print a space. OK, so just think about that. HANNAH: And then the same part of the problem set is a program called greedy.c, so that you're going to want to think about conditionals and making sure that you can make proper change. And one little warning we have for you is be careful of floating point values. If that means absolutely nothing to you, it'll be covered in lecture this week and also in [INAUDIBLE] walk through, which you guys will learn to love on the problem sets.  One thing that I really suggest, especially with Mario.c, when you're doing the problem set, if you get stuck, start by doing it on paper. Write it out and actually sit there and pretend to be the computer, and go through-- say I were the computer, how would I follow this for loop through? How would my variables in the for loop change? So doing it on paper makes it 10 times easier when you go to sit down at the computer. So just my little plug.  SHARON: And also don't think that you have to code everything all at once. Make sure you take an iterative process. Do a little bit, print it out, see what happens. Sometimes it's a little trial and error. And come to office hours. Super fun. JASON HIRSCHHORN: So any questions? All right guys, that was your first section. HANNAH: Thanks for coming. JASON HIRSCHHORN: Thanks for come. SHARON: Thank you. [APPLAUSE] | Джейсон Хіршхорн: Вітаю. Хочу зробити кілька оголошень перед початком. Щодо семінарів, з якими всі вже мали визначитися на даний момент. Семінари проходитимуть з неділі по вівторок. Отже, ви , панове маєте отримати своє завдання. А потім наступного тижня, я так думаю, ми визначимо ваших асистентів. Якщо у вас виникли певні питання або проблема, або ви забули записатися на семінари. З деякими таке трапляється. Просто повідомте нас. Пишіть на пошту організаторам курсу.  Далі щодо консультацій. Ми визначили, коли вони проходитимуть, минулого тижня. Одна консультація на тиждень. У понеділок – у Льоверетті з 8:00 до 11:00, у вівторок в Кеботі з 8:00 до 11:00, Середа, Мазер з 8:30 до 11:30. І в четвер у Анненберзі з 8:00 до 11:00.  Щодо Скретч.: багато людей не потребують консультацій для роботи у Скретч, що цілком нормально. Якщо ви потребуєте, це також абсолютно нормально. Але у майбутньому задачі ставатимуть набагато складнішими. Тож консультації стануть вам у пригоді. Обов’язково відвідуйте їх. Плануйте на них ходити. Коли ви дістанетесь до завдань 4,5, 6 і 7, вам необхідно буде відвідувати консультації. На них вам допоможуть зробити вашу роботу.  Наприклад, минулого року, консультації були у четвер - якщо завдання треба було виконати на п’ятницю. Тож у четвер ввечері у нас було по 200-250 студентів. Отже, скористайтеся цією можливістю. Консультації - ваші кращі друзі. Хоча тут ви напевно будете застрягати над завданнями, але тут вам їх і допоможуть вирішити. Отож-бо консультації. Тож це були оголошення. Оголошення зробив, можна починати.  Так, щодо віртуальної  лабораторії. Усі вже завантажили віртуальну лабораторію? Ні. Якщо так, тоді добре. Я вже завантажив. Вона нова у цьому році, тому ми напевно матимемо помилки у ній. Будь-ласка, завантажте її якомога раніше. У завданні один буде інструкція по її завантаженню. Починайте її встановлення прямо тут. Тут сказано, як завантажити віртуальну лабораторію.  Тож завантажити віртуальну лабораторію краще раніше, ніж пізніше. Інструкції знаходяться у спеціальному практичному завданні. Якщо ви дочекаєтесь вечора середи, а потім у вас виникнуть проблеми, і ви надішлете нам листа, що вам не вдається завантажити лабораторію, в середу опівночі, ви ж розумієте, що це не поважна причина задля відкладення. Ви маєте завантажити її зараз, і все одно ви матимете певні складнощі. Я намагався зробити це учора. І у мене вони виникли. Якщо складнощі виникнуть при встановленні, обов’язково повторіть спробу, тому що я невдало намагався встановити програму декілька разів, і в кінці кінців все спрацювало, хоча цьому немає пояснення, але все встановилося. Тож намагайтеся декілька разів, але якщо вже ви наштовхуєтеся на цегляну стіну, пишіть нам на пошту, і ми із задоволенням допоможемо вам, хлопці і дівчата. Так?  Студент: Просто маленьке питання. Якщо перевіряється стиль, то там сказано, що ми не включили його.  Джейсон Хіршхорн: Поки що ні.  Студент: Але тут сказано виконати це. Як треба виконувати, щоб упевнитися, що цей компонент присутній у системі?  Джейсон Хіршхорн: Треба запускати команду bash. Я повернуся до цього пізніше. Чудово. Програма може трохи лякати вас, оскільки ви звикли працювати у графічному користувацькому інтерфейсі. Що я маю на увазі, коли кажу, що Скретч був графічним інтерфейсом? Я маю на увазі, що коли ви програмуєте, ви загалом використовуєте блоки коду. Ви можете бачити код і все таке подібне.  У цій віртуальній лабораторії ви будете використовувати командний рядок. Отже у вашому програмувальному майбутньому ви будете працювати з програмами через командний рядок. Отож добре було б, якби ви занурилися у це середовище раніше і вже отримали певний досвід. Давайте перейдемо до справи. Це Роб.  Давайте подивимося на командний рядок. Ось такий вигляд має віртуальна лабораторія. Мені хтось може сказати, навіщо ми використовуємо віртуальну лабораторію? Що це таке? Хтось знає? Давайте.  Студент: Це віртуальний механізм, за допомогою якого ви запускаєте свою машину, таким чином що ви можете подолати розбіжності ОС між Маком і ПК.  Джейсон Хіршхорн: Чудово.  Студент: Це універсальний зразок.  Джейсон Хіршхорн: Так, це схоже на універсальний зразок. Чудово. Отже, це віртуальний механізм. Загалом це операційна система. Вона еквівалентна до вашого Mac, або ПК, або Linux. Це в точності те ж саме. Отже, ви можете проводити з нею ті ж самі операції. Наприклад, якщо ви хочете, то можете вийти в інтернет. Ви не бачите цього, але прямо тут є Google Chrome. Ви можете сидіти в інтернеті, якщо забажаєте. Це операційна система. Ми пояснюємо це тому, що набагато легше, коли ми даємо інструкції. А якщо це одне й те саме середовище, це набагато легше і для нас, і для вас. Ви не стикнетесь з жодними відмінностями, коли будете програмувати. Ми точно знаємо, з чим ви стикнетесь. Отже, ось вам програма.    Коли ви почнете програмувати, вам треба подивитися у лівий нижній кут. Там є невеличка іконка. Вам треба натиснути на неї. Це ваше вікно командного рядка. У ньому ви будете працювати протягом семестру. Дозвольте, я збільшу його.    Дістатися до робочого вікна не так вже й легко. По-перше, там немає іконок. Я не можу нічого натиснути. Там немає чого натискати. Тож вам необхідно буде додуматись, як це зробити, додуматись, де ви знаходитесь, що тут є, і як рухатись далі.  Тож дві найкорисніших команди напевно будуть, перша - ls - як гадаєте, що значить ls?  Студент: List (список). Джейсон Хіршхорн: так, списки. Команда просто перелистує алгоритми у каталозі. І ще cd. Скажімо, я хочу cd CS50. Що це означає? Так?  Студент: Це змінює ваш каталог. Джейсон Хіршхорн: Чудово, так. Це просто змінює каталоги. А тепер повернімося. Як мені повернутися? Наприклад, я хочу повернутися до попереднього каталогу... Так, будь-ласка.  Студент: я гадаю, вам треба написати cd просто без нічого. Джейсон Хіршхорн: Так можна зробити. Чудово. Отже якщо я напишу cd, це поверне мене назад, туди, де знаходиться тільда. Отже це поверне мене до домашнього каталогу. Припустимо, я зробив щось на кшталт цього. Наприклад, cd CS50. Знову ls. І тепер вже інша ситуація. cd super section. cd file, я занурююся глибше. А потім, скажімо, я хочу просто повернутися до початку cd. Припустимо, я не хочу робити цього поволі. Я просто хочу одразу повернутися до каталогу, який прямо наді мною. Як мені це зробити? Наприклад, cd CS50. cd super section. А я зараз отут. Я хочу повернутися за одну операцію. Що я маю зробити? Так, прошу.  Студент: Вам варто знову написати cd supersection .  Джейсон Хіршхорн: Можна це зробити. Ну я у великому розділі.  Студент: О, перед цим.  Джейсон Хіршхорн: Так, ви можете так зробити. Ви вводите cd ~CS50. Якщо ви подивитесь сюди, то це буде ваш шлях. Ось ваш шлях. Тож ви дійсно просто можете ввести cd ~/CS50. Раз - і ви там. Але який легший спосіб це зробити? Цей був абсолютно вірним, але є значно легший спосіб зробити це, тому що вам доведеться занурюватись у каталоги, і виходити з них багато разів. Тож давайте повернемось до cd supersection. Скажімо, я хочу повернутися дуже швидко. Ви можете ввести щось типу cd.. (дві крапки). Це відкине вас назад дуже швидко. Тож cd supersection. cd files Нехай я буду тут. Так ls, я не хочу бути тут більше - cd.. дві крапки. На один рівень вище. cd.. підіймає мене на рівень вище. cd.. і повертаємось до вихідного стану.  Добре, отже cd, ls. Напевно найважливіші команди, бо вам необхідно буде знати, де ви знаходитесь. ls вказуватиме вам ваше місцезнаходження, а за допомогою cd ви зможете його змінювати.  Ви також можете робити інші круті штуки. Наприклад, що стосується ls. ls просто показуватиме вам файлові каталоги. А крім того, самі файли. Але ця команда може показати вам і те, що знаходиться всередині них.  Якщо ви хочете щось більш прикольне, то ви будете робити це за кілька тижнів. ls тире l. Це видасть вам трохи більше інформації. Хтось може здогадатись, що означають drwx, rwx і штуки типу цього по ліву руку? У когось є якісь ідеї з цього приводу?  Студент: Дозволи.  Джейсон Хіршхорн: Так, довершено. Дозволи. Ви стикнетесь з цим пізніше, коли почнете програмувати. Якщо ви колись, будучи в он-лайні, натискали на зображення чи на поле, і з’являвся текст - типу доступ заборонений або дозволу не надано чи щось на кшталт цього, це тому, що коли писався сайт, вони не встановили дозволи для того, щоб ви могли це зробити. Ось звідки це все йде.  Так, чудово. Отже, ми вийшли з каталогу. Ми можемо подивитися на вміст нашого каталогу. Але як ми можемо наповнити його цим вмістом? У мене тут є каталоги. Це чудово. Як мені зробити новий каталог? Припустимо, я хочу створити новий каталог саме зараз. Ви можете зробити щось типу цього, mkdir. У вас є новий каталог. Як назвемо цей каталог? Студент: "Чудовий". Джейсон Хіршхорн: "Чудовий". Мені подобається. Зробимо каталог чудовим. А тепер давайте ви. Нам необхідно створити список. У вас є новий каталог. І ви можете сказати, що цей каталог належить до вашої програми, оскільки він має такий колір, вірно? Блакитно-фіолетовий чи як там. Давайте його змінимо. Змінимо каталог. Ми йдемо в каталог, отже, будь-ласка? Студент: А список цього всього є в інтернеті? Джейсон Хіршхорн: так, я викладу його трохи пізніше - там будуть слайди. Вони будуть в PDF-форматі. Ми викладемо їх для всіх охочих. Так, вибачте, дійсно багато команд. Якщо ви хочете, щоб я уповільнився, або ж повернувся трохи назад, або щось таке, повідомте мене про це. Це забагато, щоб осягнути за один раз. Прошу. Студент: можу я задати потенційно дурне запитання? Джейсон Хіршхорн: Дурних запитань не буває. Студент: Ось одне з них. Каталог - для того, щоб проглядати інформацію у місці її розміщення?  Джейсон Хіршхорн: так.  Студент: Чи принципово те, що ми робимо? Якщо ми створюємо щось , наприклад, hello world, чи щось подібне, воно розміщується у каталозі, так?  Джейсон Хіршхорн: Так.  Студент: Це стосується всього? Все відображається у каталозі?  Джейсон Хіршхорн: так. Що би ви не робили, у якому б каталозі ви не працювали, те, що ви робите буде відображатися у цьому каталозі. Студент: А коли ми отримати дані назад, ми отримуємо їх з каталогу?  Джейсон Хіршхорн: Так, каталог містить у собі все. Цей механізм схожий на те, коли у вас є нова папка на робочому столі, і потім ви створюєте текстовий файл і розміщуєте його у цій папці, і він там залишається. Зрозуміло?  Студент: Так.  Джейсон Хіршхорн: Так, прошу.  Студент: Я можу попросити вас трохи збільшити зображення?  Джейсон Хіршхорн: О, звичайно. Перепрошую. Проектор такий поганий. У нас виникало з ним стільки проблем раніше. Так краще? Не завелике?  Студент: Ні, ні, все добре.  Джейсон Хіршхорн: Ні, все добре. Воно не замале? Так, що ви хотіли?  Студент: Перепрошую, а яка команда для видалення?  Джейсон Хіршхорн: О, видалення. Якщо ви хочете щось видалити. просто наберіть clear. clear enter. В останній програмі, наприклад, дані не видаляються, вони просто пересуваються вниз на екрані. Тож якщо ви введете команду clear - наприклад, я прокручую вгору. Я можу прокрутити вгору. Це все, що я зробив. Але якщо я ввів clear, а потім прокрутив, то ось де це буде. Отже, програма не стирає інформацію, а просто надає вам новий майданчик.  Студент: Яким чином можна (не чутно).... каталог?  Джейсон Хіршхорн: Ми ще до цього дійдемо. Пригальмуйтеся.  [Сміх] Добре. Так, зараз ми у каталозі. Мені здається, Девід у своїй лекції показував класну штуку. Давайте напишемо програму. Ви можете зробити це різними способами. Ви можете використовувати будь-який текстовий редактор. Ви можете використовувати Nano, або Vim, чи Emacs. Це трохи складніше. Або ви можете використовувати отаку річ, яка називається Gedit. Вибачте, Gedit - це загалом звичайний текстовий редактор. Типу текстовий процесор, що редагує текст. У нього досить гарний інтерфейс, ось такий. І вам необхідно робити щось на кшталт цього. Дозвольте мені дати йому ім’я , бо я не зробив цього.  Як будуть звати нашу першу програму? Вона буде просто казати "Привіт".  Студент: [Не чутно].  Джейсон Хіршхорн: Вам не вистачає уяви. Добре, Gedit. Привіт. Я створю txt. ОК, є. Давайте напишемо програму. Я думаю, ви бачили це протягом лекції. Вам потрібна головна функція. Вуп. Це головна функція. Зробити більше? Так, вибачте. Добре, отже вам необхідна головна функція. Я гадаю, я трохи поспішаю, але ніколи не зарано звикати до таких речей.  Тож головна функція. Будь-яка програма на мові С, яку ви будете писати до кінця цього курсу, буде мати головну функцію. Хтось знає, чому так? Будь-ласка.  Студент: Початок роботи.  Джейсон Хіршхорн: Саме так. Це повідомляє вашій програмі, де запускатися. Якщо у вас немає головної функції, і ви компілюєте програму і все таке, то вона не знатиме, коли запускатись. Головна - це завжди, де запускатись. Тож ви маєте головну функцію, а хтось пам’ятає, якїї вводити?  Студент: printf. Джейсон Хіршхорн: так, printf, привіт printf. Добре, друзі, наразі я привів вас до провалу, але чи буде це компілюватися.  Студент: Ні.  Джейсон Хіршхорн: Чому? Є багато причин для цього. Звичайно. не буде.  Студент: Немає стандартного … (не чутно).  Джейсон Хіршхорн: Чудово. Яке у нього призначення у даному випадку?  Студент: [не чутно]  Джейсон Хіршхорн: Це файл заголовка, вірно? Це файл заголовка. І ви абсолютно праві. обидва з вас мають рацію. І щоб включити це, я просто введу команду, щоб включити стандартний вхід-вихід. .h. Зараз можна запускати? Вона компілюватиметься?  Студент: ні. Джейсон Хіршхорн: чому ні? Студент: Це [INAUDIBLE] текст. Джейсон Хіршхорн: так, прекрасно. Тож я написав програму. Я зберіг її, але gedit hi.txt. Тепер якщо я піду до ls, ось вона. Ось тут. Вона не в папці Чудова. Я напевно вийшов. О, чудово. Як нам помістити це у Чудову?Я не хочу її у своєму домашньому каталозі разом з усім іншим. Я хочу помістити її у папку. Як мені це зробити?  Це не надто складно. Переміщатись. Ми вже вивчили ls, список. cd, щоб змінювати напрям. cd.., щоби виходити з каталогів. А тепер нам треба вивчити навігацію між файлами. Треба просто пересунути hi.txt, і я хочу помістити його у «Чудову».  Але обережно. Коли б ви не писали у командому рядку, все має значення. Прописні літери мають значення. Якщо я напишу рядкову а, програма не розумітиме, що відбувається. Все грає роль, тож будьте дуже конкретними.  Запускаємо цей список. Його тут немає. Давайте змінимо назву каталогу на «Чудова». ls. Бум, і він з’являєтсья.А хтось зрозумів, чому це не буде компілюватися? Чому знову не буде компілюватися? Так?  Студент: це не в a .c.  Джейсон Хіршхорн: Довершено. Це не a .c., тож як мені перйменувати це в a .c.? Як це легко зробити? Давайте використаємо команду, яку ми тільки що вводили. Тож ми перетворимо hi.txt на hi.c. ls hi.c. А тепер просто для розваги, давайте потрапимо в нього дуже швидко. hi.c. Ви помітите, що він гарний і барвистий. Це просто допоміжні кольори у програмуванні. int і void - це типи даних. Ми розберемо це за кілька хвилин. "Привіт" - це рядок. Але корисні кольори. Якщо ви колись помилково обирали невірне розширення, у вас мали з’являтися ці кольори. Так?  Студент: Як я можу скопіювати у файл [не чутно] у інший файл? Джейсон Хіршхорн: О, я також покажу це вам. Те, що видаляє копію знаходиться в dock. Давайте спробуємо. Переконаймося, що вона працює. Напишіть "привіт". Працює, запускайте. Отже ./hi awesome.. ЦЕ ваша перша програма. Я збільшу. Дійсно дуже добре - так, прошу. Студент: Чому коли я написав gedit, не з’явилося нового рядка для gedit hi.txt? Просто курсор перемістився у рядок. Студент: так. [не чутно].  Джейсон Хіршхорн: стривайте, поясніть ще раз.  Студент: коли я набрав gedit перший раз, у першому рядку.. Джейсон Хіршхорн: угорі? Студент: там немає програми J Harvard, а курсор просто переміщується до кінця у лівий бік. Джейсон Хіпшхорн: до кінця у лівий бік? Студент: так. У когось є щось подібне?  Джейсон Хіршхорн: о, J Harvard це просто користувацьке ім’я. Це просто стандартне ім’я користувача.    Студент: воно перестає відповідати. Джейсон Хіршхорн: о, перестає відповідати. Добре, що ви маєте зробити - так це закрити Gedit. Перезапустіть Gedit, і має повернутися. Gedit зараз запущений на вашому пристрої? Якщо ви перезапустите його, усе має стати на місця. Спрацювало?    Студент: ви використовували команду gedit , щоб відкрити.. Джейсон Хіршхорн: ні, перша команда gedit була помилковою. Першою командою у gedit я створив gedit hi.txt. Тож я назвав його gedit hi.txt. hi.txt - це ім я файла. Перша команда була помилковою, а потім я швидко закрив редактор. Тож питання у тому, якщо я користуюся Gedit, то я не зможу нічого робити у командному рядку, поки не закрию Gedit, тож я гадаю, з цим ви стикнулися. Просто закрийте Gedit. Це поверне ту програму і ви знову матимете змогу писати у командному рядку. Зрозуміло? Клас. Просто переконайтеся, що у вас все закрито, якщо виникають проблеми.  Добре, ми розібрали тільки що дуже важливе питання. Тепер у мене є hi.c. Але скажімо, я хочу змінити його. Я хочу значно прокачати себе. Хочу сказати "привіт, клас". Але я хочу зберегти цей шаблон. Немає необхідності переписувати цілу програму з нуля. Я хочу використати шаблон. Як мені копіювати?  Легко. cp hi.c newhi.c. Тепер у мене є дві програми. А тепер я можу відкрити newhi.c, і замість "Привіт", я напишу "Привіт, клас"! Я повертаюся. Виходжу. Повертаюся до командного рядка. У ls створюю newhi. Запускаю його. Так. Тепер це краще, набагато краще. Якщо ви хочете копіювати щось настільки ж легко, то cp - це копіювати.  Таким чином ми розібралися з переміщеннями, створеннями каталогів і копіюванням. Давайте подивимось, що ще я маю розібрати? О, так. Давайте подивимось ще це. Тож це «Чудова». Скажімо, я хотів - а , постривайте. Я повернусь назад у «Чудову».  Скажімо, я не хочу, щоб ці файли були тут. Наприклад, я хочу перемістити їх у новий каталог типу каталогу «Чудовий». Я хочу почати створювати його, і тепер у мене є новий каталог під назвою hi. У ньому вмістяться всі програми, які я написав під назвою hi. Як можна це зробити?  Студент: змінивши каталог  Джейсон Хіршхорн: так, hi. Вуп. mkdir hi\_dir. Воно скаржиться, бо я вже створював тут hi. Тепер у мене є каталог hi. Тепер я хочу перемістити усі програми, що я написав, у каталог hi. Прибратися. Як мені це зробити?  Студент: [не чутно].    Джейсон Хіршхорн: перемістити їх, так? Давайте спробуємо. Так?  Студент: у чому полягає різниця між hi та hi.c, який у нас вже є. Джейсон Хіршхорн: о, hi.c первинний. Це файл, який ви написали. Якщо б ви відкрили його у Gedit чи Nano, це те, у чому ви писали би. Після компіляції, ви отримуєте виконуваний файл. Це hi. Якщо ви відкриєте hi, це буде якась купа маячні. ЦЕ загалом купа комп’ютерних інструкцій. Так, прошу.  Студент: що означають кольори? Блакитний - це напевне каталог. А що таке жовтий?  Джейсон Хіршхорн: це виконувані файли. Коли я створив hi.c, це видало мені -- тож я можу зробити це. Забігаючи наперед. Видаляємо hi. Тепер давайте ls. Бачите, більше немає hi. Коли ви так робите, це значить, що ви берете написану вами програму і створюєте виконуваний файл. Тож створіть hi.c -- тепер створіть hi. Я думаю, ви розглядали це на лекції.  Це купа компіляторських інструкцій. Тож замість того, щоб бути змушеним морочитися з оцим всим. Це прапорці. Ви просто пишете форму. За рахунок цього створюються виконувані файли. Коли ви вводите ls, у вас є "привіт". Тож це те, що ви запустите. І ви запускаєте програму командою ./hi, і вона починає працювати. Але якщо я спробую зробити це з hi.c, ні, це не працюватиме. Так, це має сенс?  Студент : угу.  Джейсон Хіршхорн: Круто. Стривайте, я хотів сказати дещо інше. Дуже швидко, про всяк випадок. якщо когось це цікавить, коли ви виходите з каталогу - це крапка крапка. У когось є ідеї , що значить одна крапка? Студент: [не чутно] Джейсон Хіршхорн: це взагалі поточний каталог. Тож це поточний каталог. Можливо, це зайве казати, але ви маєте зробити так ./ і ім’я вашої пограми, щоб ваша програма запустилася. Ви маєте так зробити. Добре.  Студент: якщо ви напишете cd і три крапки, невже ви підніметесь на стільки каталогів вгору?  Джейсон Хіршхорн: Ви маєте на увазі cd крапка крапка крапка крапка? Студент: так. Джейсон Хіршхорн: Ні. Пам ятаєте, я задавав команду ls -l? Якщо я роблю щось інше, то треба ls -a. Тобто, -a показує усе. Вона показує вам приховані та неприховані речі. Тож якщо ви були уважними, то ви помітили, що коли я набираю ls, я просто бачу те, що я зробив. Якщо я додаю -a, я бачу все. І в тому числі поточний каталог і каталог над ним. Зрозуміло? Так?  Студент: але, власне, це не у каталозі. Поточний каталог не знаходиться у собі, тож чому треба використовувати крапки усередині?  Джейсон Хіршхорн: Ви маєте на увазі, що це типу поза каталогом?  Студент: саме так. У нас є список речей, які не знаходяться у каталозі.  Джейсон Хіршхорн: Я думаю, що вони у каталозі. Так як -a показує все, навіть якщо воно приховано у каталозі. Тож я гадаю, що це просто шлях до поточного каталогу, якщо це має сенс. Це маленька рекурсія. Як у Початку. Так, прошу.  Студент: тож ви кажете, що -a показує [не чутно]? Джейсон Хіршхорн: -a, саме так.  Студент: Що робить це [ не чутно]? Джейсон Хіршхорн: Ну наприклад, давайте повернемось нагору. Це такі штуки, у яких ви не повинні плутатись і ви не маєте хвилюватись за це. Це типу домашня сторінка. Якщо я задам команду -a, я отримаю тонни всього. Взагалі все. Вам не треба возитися з оцим всім. Не починайте заглиблюватись, і просто видаляйте ці дані.  [Сміх]  Так, це були більш звичні речі, але про це варто знати. Я гадаю, що варто. Але якщо ви ніколи не хочете задавати команду -a протягом цього курсу, просто забудьте про неї. Так, прошу.    Студент: Що таке ls -l? Джейсон Хіршхорн: Ця команда показує вам дозволи. Загалом ви використовуєте її для дозволів, отож -l. Ця команда показує вам трохи більше інформації, але знову ж таки, це вже більш звичні штуки. Це просто дозволи. Дозволи для папок. Скажімо, я зробив папку, тож d означає каталог. Це каталоги. А потім читайте, пишіть, виконуйте. Це дозволи, я гадаю, для користувача, групи, світу. Але ми зайшли занадто далеко. Ви повернетесь до цього за кілька тижнів, але просто , щоб ви знали, що це дозволи. Так, що? Студент: Ви можете очистити перш ніж перейти до наступних речей, можна так? Джейсон Хіршхорн: ви маєте на увазі, видалити? Не знаю. Ніколи так не роблю. А чому ви хочете так зробити? Навіщо? Мені подобається. Ви типу живете небезпечно. Не хочу нічого запропонованого. Це круто. Студент: [не чутно]. Джейсон Хіршхорн: так, ми можемо це зробити. Перепрошую, порання з розміром лякає мене наразі. Давайте рухатись далі. Так, тепер ви можете рухатись далі. Все, чому я навчив вас, залишиться назавжди. Дякую. Ми створюємо наші каталоги. Ми створюємо файли і штуки типу цього, тож у нас досі все це є. Добре. далі рухатимусь швидше.  Так, давайте дійсно швидко. Скажімо, я у «Чудовій». Наприклад, я хочу перемістити усе до нового каталогу, тому що я хочу організувати все, так? Тож як мені зробити це. Перемістити. Ми використаємо команду move знову. Отож move hi.c. І що ви робите, так це пролистуєте усе, і те, що останнє у списку вказує на місце, куди ви його перемістили. Тож move hi.c, hi newhi newhi.c, hi\_dir. І все перемістилося у останньому напрямі, котрий ви зазначили. Далі змінюємо каталог , hi\_dir ls. Усе там. Це гарно і більш організовано.  Добре, скажімо, Я ненавиджу мою первинну hi-програму. Я хочу здихатись її. Як мені це зробити? Так?    Студент: [не чутно]. Джейсон Хіршхорн: Припустимо, я просто хочу видалити цю програму, так що просто hi. Ні, ви маєте рацію. Так, це rm, але ви вводите rm hi. З’явиться невеличке попередження. Там буде запитуватися, чи ви впевнені, що хочете видалити цю програму. Якщо ви не хочете таких попереджень, якщо у вас обмаль часу, то видаліть -f. Це інший прапорець. Це як -l. Це як -a. Це просто відмітки, якими ви відмічаєте ці команди. Тож видаліть -f. -f значить примусово.  Тож приберіть це "примусово". Я не хочу, щоб мене попереджали. Отож видаліть -f Давайте зробимо hi.c. Просто позбавтесь від цього. Бачите, попередження не було.  Але скажімо, мене дістав цей каталог. Я хочу позбавитись від нього. Тоді я задаю команду remove hi\_dir. Ні, це не працює. Так я не зможу видалити каталог. У когось є ідеї?  Студент: Може це remove dir? Джейсон Хіршхорн: Типу так?  Студент: ні, просто [не чутно]. Джейсон Хіршхорн: ні, зараз там немає hi. Це просто hi\_dir. Це каталог. Тож загалом , що ви хочете хробити - о у вас є ідея? Я бачу півруки.  Студент: [не чутно]. Джейсон Хіршхорн: Добре.  Студент: [не чутно].  Джейсон Хіршхорн: Добре, тож, якщо я хочу видалити каталог, то видалити -r. Це рекурсивно. Тож видалити цей каталог - можна рекурсивно. Я хочу, щоб ви зайшли у цей каталог, видалили все з нього, а потім позбавилися від самого каталогу. А потім я також задам команду f, тому що я не хочу отримувати попередження кожного разу, як я видаляю файл. Отож видалити -rf hi\_dir. Бачите, він зник. Будьте обачні, коли ви використовуєте це. Я збираюся використовувати цю команду тут також. -rf Чудову. Вона зникла. Будьте обачні при використанні. Якщо ви використовуєте її тут, і набираєте щось типу цього, ніколи так не робіть. Е видалить всі дані вашої програми, навіть не спитавши у вас. А я це зробив. Сортуючи завдання у 4:00, я намагався видалити каталоги з цими завданнями, і я зробив це. І це було О Боже мій! Не робіть так - я намагався скасувати операцію. Ви можете скасувати. Команда C скасовує усе, але я вже видалив 75 відсотків. Тож я налажав. І інші так роблять. З усіма трапляється. Але не робіть цього у цьому семестрі. Це фігня. Я нервую, коли воно тут. Краще видалю.  [Сміх] Можете уявити собі? Добре, питання. Так?  Студент: Якщо я зроблю усе, що ви робили тут, це те ж саме, що зберегти файл як [ не чутно]. Джейсон Хіршхорн: ви маєте на увазі через Gedit. Студент: так. Джейсон Хіршхорн: саме так. Gedit корисний. Він дуже корисний. Протягом курсу ви можете робити багато речей через Gedit. Ви можете зберігати. Ви можете перейменовувати. Ви можете переміщувати файли і щось подібне, але, чесно кажучи, коли ви наблизитесь до 50 --, коли будете у 50--51, якщо заглибитесь у 61, це стане вашим життям. Це командний рядок. Це те місце, де ви будете писати команди до кінця свого життя, тож напевне використовуйте Gedit, якщо є бажання. Це дуже хороший спосіб для початку. Для перших практичних завдань напевне використовуйте його, але з часом привчайте себе до командного рядка. Але, звичайно. ви можете. Так?  Студент: Тож коли ми хочемо видалити файли, які ми створили, є два способи. [не чутно]. Є спосіб, коли ви видаляєте каталог, щоб поставити попередження, і щоб видалення не відбулося? Джейсон Хіршхорн: rm -r. Типу ви не змушуєте його. Тож rm -r. Це буде відбуватися рекурсивно, видаляти все і попереджувати вас стосовно всього, що буде зводити вас з розуму. Я зазвичай роблю тільки rm -f та rm -rf, постійно. У мене просто немає часу. Я зайнята людина. Добре, прошу? Так.  Студент: там є місце для видалених файлів чи щось на кшталт корзини? Джейсон Хіршхорн: Ні. [Сміх] Чи я можу повернутись і отримати назад програму, яку я щойно видалив, ні? Ні, ні і ще раз ні. Якщо ви задаєте команду rm, програма зникне. Так?  Студент: Тож ви можете [ не чутно] разом. Джейсон Хіршхорн: так, типу того. Так, наприклад, ви бачили rm -rf. Скажімо я хочу зробити ls -la, це покаже мені усе, навіть приховані речі з їх дозволами.  Студент: Тож потім r, -r flag, якщо ми так не зробимо, ми не можемо видалити каталог, вірно?  Джейсон Хіршхорн: саме так.  Студент: Тож це вказує нам на те, що ми хочемо зробити щось із каталогом? Таке у нього значення?  Джейсон Хіршхорн: -r означає рекурсивно. У каталозі знаходяться файли, правильно? Тож ви хочете потрапити у цей каталог, видалити усе, всі файли з каталогу, потім вийти і видалити власне каталог. Тож -r означає рекурсивно, якщо це має сенс. Так.  Студент: Тож каталоги треба [не чутно] і просто видалити. Чи ще треба--  Джейсон Хіршхорн: Давайте спробуємо. Тож mkdir пустий. Видаляємо пустий каталог. Ні.  Студент: ні, вам ще треба --  Джейсон Хіршхорн: так, завжди. Отож -r, -rf, пустий. Чудово. Ще питання? Я думаю, це ваш експрес-курс з командного рядка. Ще питання, бо ми маємо перейти до типів даних, і циклів, і всього такого. Так?  Студент: у мене CS50 не показується як один з моїх каталогів. Це нормально?  Джейсон Хіршхорн: о, так. Це так у мене. Так, тут у мене просто вирішення ваших завдань.  [Сміх]  Так, це абсолютно нормально. Я думаю, у вас напевне є ці чотири. Це так і має бути. Так, добре, ще питання? Я знаю, що це було ультра-швидко. Прошу.  Студент: я помітив колір морської хвилі. Що це?  Джейсон Хіршхорн: ЦЕекаталог. Студент: А цей блакитний?  Джейсон Хіршхорн: о, ви маєте на увазі цей колір морської хвилі?  Студент: ні.  Джейсон Хіршхорн: цей?  Студент: ні, коли ви пролистували-- Джейсон Хіршхорн: о. Цей?  Студент: так, іконки.  Джейсон Хіршхорн: о, ось. Я не знаю. Що це означає? Це схоже на вказівник до каталогу.  Студент: це схоже на псевдонім.  Джейсон Хіршхорн: Можливо. Вам не треба цього знати. Ще питання? Чудово. Якщо у вас є якісь питання, я знаю, що це було забагато. Багато інформації буде викладено у мережу. Я викладу це для вас, друзі. Семінар B (другий семінар) також буде онлайн. Якщо ще залишилися якісь питання, просто підійдіть пізніше і задайте їх мені. Так, все. Тепер Шерон і Ханна пояснять логіку практичного завдання номер один.  Шерон: Дідько, у нас проблеми.  Ханна: Тим часом, я Ханна. Це Шерон. Ми обидві - організатори курсу CS50 , і ми плануємо дати невеличке інтро щодо деяких важливих речей для практичних завдань. Все від типів даних до умов для циклів. Зациклювання загалом.  Шерон: А у кінці ми розберемо перше практичне завдання, і що ви маєте знати для цього. Давайте почнемо з типів даних. Ви, друзі, маєте бути знайомі з тим, що виділено блакитним. Тож ми можемо почати з ints. ints - це цілі числа, отже один, два, три, чотири. float - це цілі числа з плаваючою комою типу 5,2 чи навіть 5,0. Chars - це літери на кшталт A, B, C. А string, ви усі знаєте. Це типу CS50 чи hello та world.  Bool - це логічний вираз, отож якщо у нас є значення істинно чи хибно, то це згідно з логікою. Тож якщо щось будеістинним, то 5 дорівнюватиме 5, а якщо 5 дорівнюватиме 4 - це невірно, тож це вважається хибним. А ось порівняльний розмір всіх цих типів даних. І вам треба буде вивчити це для тестів, тож це варто запам’ятати, але ми опублікуємо це. Взагалі, вже опублікували. Вам має бути знайомо те, що виділено блакитним.  Джейсон Хіршхорн: дуже швидко, просто якщо ви хочете дізнатися різницю між char і string. Якщо ви пишете щось, навіть щось типу "hi,", це буде рядок (string). Якщо є лапки, то це рядок. Якщо я напишу просто "h" - це все ще рядок. Але якщо буде лише одна лапка, це char (символ). І ви можете спитати , у чому різниця. Різниця у тому, що це 1 байт пам’яті. А це 4.  Шерон: float? Пояснити знов? Добре, це число з десятковим дробом, тож 5.2 чи навіть 5.0 - це float, а 5 - це int. Студент: [не чутно].  Шерон: о, може бути довгий дріб типу 5.675 чи 3.1415. Так?  Студент: Чи ми маємо знати всі розміри для цього? Шерон: так. Студент: [не чутно].  Шерон: про білі поки що не хвилюйтеся.  Ханна: записи ми виклали на CS50.net/sections. Студент: Просто для задоволення цікавості, чи є якась програма для розпізнавання розмірів файлів, про які ви кажете, що ми маємо їх знати?  Шерон: Це типи даних, але так. У цьому випадку який займатиме більше пам’яті чи більше простору?  Джейсон Хіршхорн: Якщо на конкретному прикладі, то я гадаю, що у четвертому чи п’ятому практичному завданні ми дамо вам купу даних і вам доведеться переводити ці типи даних з одного в інший. Для цього корисно знати, що якщо у вас стільки цілих чисел, то що це буде. Скажімо, я дав вам 10 цілих, який у них буде сукупний розмір? Студент: 40. Джейсон Хіршхорн: 40 байт, дійсно. Тож ви маєте для себе знати, що вам потрібно буде подолати 40 байтів. Припустимо, замість цього я дав вам 40 символів, тоді-- скажімо, я дав вам 10 символів, тоді ви знаєте, що вам треба перестрибнути 10 байтів. Дуже корисно знати розміри типів даних, тому що ви будете переміщуватися серед них багато разів, тож ви знатимете , наскільки далеко переміщуватися.  Студент: Яка різниця між двома лапками і одною?  Джейсон Хіршхорн: Дві лапки - це рядок. Одна - це символ.  Студент: операційно у чому полягає різниця? У обох випадках є hi. Джейсон Хіршхорн: о, перепрошую. Це h. Вибачте, це не ясно.  Студент: те, що вгорі-- Джейсон Хіршхорн: це рядок, так. Це два символи, вірно? Рядок hi. Але скажімо, у мене є один символ, якщо я поставлю його у лапки, то це рядок. Це рядок h. Зрозуміло? Це чотири байта. Але якщо я позбавився від однієї лапки, і залишив лише одну. Це тепер символ, символ h. Та ж сама штука, різні типи даних. Це лише один байт, тоді як рядок - це чотири байта. Так. Студент: тож якщо ви писали би слово, наприклад «Девід», ви надали б перевагу використанню рядка, тому що незалежно від того, наскільки він довгий, він займе всього 4 байта, якщо він не подвійний, у такому разі-- тож якщо він виходить за межі чотирьох символів, то ви хочете зробити його рядком. Це загальна логіка? Джейсон Хіршхорн: якщо це більше одного символу, то це рядок.  Студент: Тож немає сенсу зберігати слово з чотирьох літер окремо по літерам чи слово з трьох літер як три окремі літери, тому що це буде три байта, на противагу рядку. Джейсон Хіршхорн: ви можете так зробити. Ви можете так зробити, але я не знаю, чи варто це робити, тому що ви просто зберігаєте один байт. Розумієте про що я кажу? У великій схемі, це мало що значитиме. Але наприклад, printf, якщо ви роздруковуєте файл, і вас тільки три літери, які ви можете роздрукувати - скажімо К-І-Т, ви можете роздрукувати це по символу, чи роздрукувати «кота» рядком. Це власне одне й те саме. Ви можете робити так, однак--  Студент: це не зекономить місця.  Джейсон Хіршхорн: так. Таке порання того не варте. Так?  Студент: наприклад, у вас є h з однією або двома лапками. Навіщо вам зберігати одну літеру як рядок? З якою метою це робиться? Джейсон Хіршхорн: не знаю. Я маю на увазі, це дійсно залежить від програми. Я гадаю, що пізніше ви будете писати коди у командному рядку. Припустимо, замість мене, програма питає вас про цілі числа -- перепрошую, можу я відповісти за тебе? Ханна: так.    Джейсон Хіршхорн: Тож наприклад, замість мене, який запитує у вас щось з get end, який ви напевне розглядали на лекції, може опинитися командний рядок. Ви просто набираєте команду у командному рядку. Це буде типу -/hello4 чи щось таке, чи hello h, неважливо. Ви робите це у командному рядку, замість того, щоб вас про це питали, цей командний рядок - це завжди рядок, чи це 4 чи h - це завжди рядок. Тож це один приклад того, коли окрема літера чи число можуть бути представлені у вигляді рядка. Шерон: Добре. А тепер деякі базові оператори. Сподіваюся, ви знайомі з першими чотирма з них у загальному вигляді. Це додавання, віднімання, множення і ділення, і переконайтеся, що ви використовуєте вірні комбінації клавіш.  Також є модуль, з яким деякі з вас напевно не дуже знайомі. А що робить модуль так це якщо ми візьмемо цей приклад 4 % 2, він показує остачу, яка залишилася після ділення. Якщо 4 поділити на 2 - буде 2, і там немає остачі. 4 поділити на 3 це 1 і 1/3, і отже остача 1. Тож воно рахує до 1. а 4 % 5 - це дріб 4/5 і остача 4. Зрозуміло? Добре, круто. І ми також розберемо тут PEMDAS ( пріоритетність операцій).  Деякі логічні вирази. Тож ви бачили два знаки дорівнює для того, щоб порівняти, скажімо, два числа. Отже, 5==5, це вважається істиною. Переконайтесь, що знак рівності лише один. Для невірних значень, їх більше або немає зовсім.  Ханна: два знаки рівності для порівняння.  Шерон: о, так. Тобто. Переконайтесь, що не один знак рівності. А не для порівнянь - це багацько. А ще більше ніж та менше ніж. Тепер логічне "та" та логічне "або", ось вони. А що воно робить так це , якщо ви пам’ятаєте в Скретч, коли виникала ця перешкода з "і", а потім ви могли сумістити два різних шматки там, ось що воно робить. І переконайтесь, що ви зробили два амперсанди. А цю кнопку ви можете знайти справа, під кнопкою Delete. Питання?  Студент: Що було [ не чутно]?  Шерон: ок, якщо ви ставите тільки одне, ви порівнюєте біти. Якщо ви пам’ятаєте, то біти - це одиниці і нулі, а це порівняння чогось іншого. Тож зараз ми фокусуємось на цьому.  Джейсон Хіршхорн: Отже, один знак рівності - це присвоювання значення, так? Типу int x=4. Тобто, змінна х = 4. тож один знак рівності - це призначення, але «і» та «або», як сказала Шерон - це побітові оператори. Вам не треба за це хвилюватися.  Студент: Коли використовувати логічне «і»та логічне «або»? Я забув, де ви це робили.  Ханна: Звичайно, якщо ви хочете перевірити чи два твердження є істинними, наприклад, скажімо, я хочу перевірити, що число знаходиться між 5 і 15, тобто переконатися, що число більше 5. Що число більше 5 і менше 15. Щоб оцінити це тверджння як істинне, вам необхідно, щоб дві його частини були істинними. З «або» вам необхідно тільки одне істинне твердження. Студент: Чудово, дякую.  Шерон: І також у Скретч, я впевнена, у багатьох проектах, ви мали справу з подібними штуками. Ханна: добре, чому нас цікавлять усі ці логічні оператори? У нас є структури, які, як ви бачили у Скретч, називаються умовами. І умовними є будь-які структури, якщо якась умова або логічний оператор є вірними, тоді треба писати код у фігурних дужках. Тут справа ви можете бачити блок Скретч. Усе, що йде у такій маленькій формі , яка виглядає ось так-- не знаю, як її назвати-- це логічний оператор чи умова.  Тож знову логічні вирази чи умови - це будь-що , що оцінюється як істина, чи хиба. І знову ж таки ви можете комбінувати логічні вирази. Ви знову ж таки можете мати х більше 5, і х менше 15. Або ви можете мати лише один з цих х, що менший від 5.  Добре, тож в С - це зліва. Це як ключове слово "якщо". У круглих дужках умова чи логічний вираз. А код, що стоїть у фігурних дужках буде просто виконуватися, буде виконуватися, якщо ця логічна умова істинна. Це має сенс? Чудово. Так. І потім як ми бачили у Скретч, ми також можемо додати інший вираз , це буде дещо у фігурних дужках, що буде виконуватися лише у тому разі, якщо умова хибна. Зрозуміло? Якісь питання з цього приводу? Чудово.  Круто. Тепер маленький приклад. Припустимо, ми хочемо визначити, в залежності від часу, що нам казати "добрий ранок" чи "добрий вечір". Я скажу, що до 12 дня, ми казатимемо "добрий ранок", а в іншому випадку - "добрий вечір". І коли я промовила оце "в іншому випадку" у С це еквівалентно до цього Іншого. Тож ми перевіряємо, і якщо воєнний час менше 12, ми кажемо "добрий ранок". У іншому випадку - "добрий вечір".  Студент: то ж ми ми вводимо цей час у командний рядок. Ханна: вірно. Чи ми ще маємо кудись ввести цю інформацію? Тут вона нічого не варта. Ми навіть не заявляли її. Я припускаю, що десь над цією програмою я заявила час, або спитала час, чи -- щось на кшталт того? Чудово.  Добре. Тепер у додаток до цього if else ("якщо інше" )формату, ми маємо дві інші структури. По-перше, маємо вмикаючу фразу, ось її загальна форма. У вас є ключове слово Switch так само як є ключове слово if. А також вводні дані, у цьому випадку n. Тож це може працювати лише з цілими. Тож це ввідне n має бути цілим числом, зрозуміло? І ми розберемо один із цих випадків, у залежності від числового значення n.  Так що в цьому випадку, ви спочатку порівнюєте чи n дорівнює першій константі, якщо так, то ви робите усе, що треба тут. Я не можу вказати на це, бо я занизька. Якщо n не дорівнює першій константі, але дорівнює другій, то ми йдемо до другого блоку, і ми можемо робити це стільки раз, скільки нам заманеться, а потім виконується за замовчуванням, якщо воно не співпадає зі значенням жодної з констант, для яких виконується цей код. Є якісь питання з цього приводу? Це трохи складніше. Так?  Студент: Що означає пауза\вихід?  Ханна: Звичайно, тож коли ми заходимо до блоку, якщо ми виявляємо, скажімо, що наше n дорівнює першій константі, ми вводимо цей блок після двокрапки. Ми робимо що завгодно на цьому рядку для коментарів, а потім зупиняємо, що означає, що ми повністю виходимо з перемикача. Студент: Зрозуміло.  Ханна: Угу?  Студент: Чи потрібен відступ, чи треба [не чутно]? Ханна: Звичайно. Отож, як завжди, відступ не обов’язково потрібний. Комп’ютеру байдуже, чи ви відступаєте. Нам, як людям, що читають ваш код, не все одно. Тож набагато легше подивитись на екран і побачити, о, я точно знаю, що відбувається, коли n дорівнює першій константі. Якби у мене були випадкові відступи, це було б не так легко. Так?  Студент: Так якщо користувач має обрати один з варіантів із списку, ми використаємо, я гадаю, [не чутно], але отримаємо ціле число з них і надамо це значення n? Ханна: саме так. Скажімо, ми хочемо перевірити, що за ціле число. Ми говоримо, о, введіть ціле число між 0 і 5, наприклад. Ми можемо спитати, яким тоді буде n, дізнатись його значення, і потім у нас буде кожен з цих випадків. І знову ж таки це те, що ми можемо дуже легко зробити з операторами, вірно? Вони можуть дорівнювати першому випадку, другому випадку, третьому випадку, а потім будь-якому з цих випадків. Це трохи швидше, але все ж трохи чистіше. Тепер це схоже на гарну структуру. Студент: Швидше ніж під час письма? Чи просто це запускається на комп’ютері трохи швидше?  Ханна: на комп’ютері. Так?  Студент: Подвійне тире (подвійний слеш) - це просто коментар чи --  Ханна: о, перепрошую. Я не вказала на це. Добре, якщо ви бачите таке // - це коментар. Просто // - коментар. Тож коли комп’ютер стикається з таким він каже "Добре, усе , що слідує за косими лініями, я буду ігнорувати. Я зроблю вигляд, що ви ніколи не писали цього".  Студент: Тож це [не чутно].  Ханна: це не робить абсолютно нічого. Але якщо я написала у першому коментарі замість printf, скажімо, "вітання, ви ввели першу константу", то від цього щось буде. Так?  Студент: Тож у реальності випадок просто буде числом а потім [не чутно] буде іншим цілим числом.  Ханна: точно.  Студент: а еліпси потрібні?  Ханна: Вибачте?  Студент: ну оце потрібно?  Ханна: о ні, перепрошую. Це просто була вказівка на те, що ви можете розглядати стільки випадків, скільки вам завгодно. Давайте розглянемо конкретний приклад, який допоможе прояснити деякі штуки. Добре, тож , наприклад, я кажу, видайте мені цілу n, яка представляє клас чисел, особливо клас чисел комп’ютерної науки. Якщо ви видасте мені 50, я скажу - дивовижно. CS50 - це вступ до комп’ютерної науки, а потім я перемикну. Це означає, що я виходжу з перемикача, тож тепер я пишу працюючий код, зрозуміло?  Якщо ви замість цього дасте мені 51, я надрукую іншу умову. А якщо ви дасте мені число, що не дорівнює 50 чи 51, то я скажу, вибачте, я не знайомий з таким класом. Так?  Студент: у вас нема зупинки  Ханна: перепрошую, у мене немає зупинки?  Студент: якщо не.  Ханна: а, якщо у вас немає зупинки? Чудове запитання. Тож що трапилося б, якби ви увійшли, і перевірили б, чи я дорівнюю 50? І скажімо, ви дорівнюєте 50, ви вводите умову. А потім ви продовжите виконувати , тож ви спитаєте, чи я дорівнюю 51? І продовжите і прорахуєте кожний із шляхів? Так?  Студент: лінія за замовчуванням рівносильна функції "інше"? Ханна: Точно. Дуже добре. Це типу розглянути все.  Студент: Якщо у вас немає зупинкиі одна з умов виявилася вірною, а потім сказано збільшити n на 1, тоді воно автоматично видає наступну відповідь. Воно перевірить випадок з 51, і потім покаже це також? Ханна: Так, я гадаю, це працюватиме. Ви можете заплутатись, тож зупинка- це корисна річ. Так? Студент: Без зупинки виконається функція за замовчуванням? Ханна: Це гарне питання. Джейсон Хіршхорн: Я гадаю, просто виконується далі. Тож якщо зупинки немає - скажімо, я ввів 50 і програма перевірила 50, і вивела дані. Роздрукування CS50 як вступ до комп’ютерних наук. У вас немає зупинки, але програма продовжуватиме виконуватися. Тож якщо зупинки немає, вона виконуватиметься. Вона виводитиме інші результати.  Ханна: Я гадаю, це виконуватиметься за замовчуванням.  Джейсон Хіршхорн: Так.  Ханна: Гарне питання. Прошу. Студент: умова є функцією? Ханна: Вибачте? Студент: умова вважається функцією? Ханна: чи умова вважається функцією. Я сумніваюся.  Джейсон Хіршхорн: зазвичай коли ви стикаєтесь з функціями, вони будуть у фігурних дужках. Наприклад, коли ви дивитесь на ваш код, наприклад, основний. Отже, кругла дужка відкривається, потім пробіл, потім дужка закривається. Функції -- зазвичай у круглих дужках. Коли немає жодних дужок - це вказівка на те, що немає функцій. Але це не так. Так?  Студент: а switch - це функція?  Ханна: чи switch - це функція.  Студент: якщо ви щось вводите, програма видає вам одну з умов. Джейсон Хіршхорн: Я б не назвав це функцією. Не усе в круглих дужках - це функція. Я маю на увазі, що це підказка, що це функція. Я б не сказав, що switch - функція, тому що насправді воно нічого не повертає. Ви потім зрозумієте.  Ханна: так, просто думайте про це як про структуру. Студент: а можна вкласти це в у функцію умови? Ханна: так, можна вкласти все, що завгодно.  Студент: Баррі сказав, що лапки виконують функцію аналогічну функції "інше". Можна замінити лапки на "інше"?  Ханна: не в цій конкретній структурі, тому що умова switch чекає на слово default. Комп’ютер знає, що default означає дещо особливе, це означає перевірити все. Щось іще стосовно цього? У нас є ще одне питання на розгляд. Так?  Студент: [не чутно] Ханна: Різниця між використанням крапки з комою і двокрапки. крапка з комою завжди каже комп’ютеру - я впоралася з цим рядком. Ви можете продовжити і запустити його. Це завершений рядок. Двокрапка у цьому випадку переміщує вас у конкретний блок. Тож крапки з комами завжди використовуються у останніх рядках. Двокрапки також використовуються для ряду інших цілей.  Студент: це еквівалентно тому, щоб сказати, якщо n дорівнює 50, тоді бла бла бла. Ханна: угу. Студент: це теж саме що [не чутно]. Ханна: саме так. І ми сказали, що вона викнонується трохи швидше. Якщо немає зупинки, це буде як якщо якщо якщо. Якщо є зупинка, - це як якщо інше якщо інше якщо інше якщо. Круто?  Джейсон Хіршхорн: Коли ви пишете код, можете просто використовувати якщо інше, але я гадаю, минулого року на тесті, треба було написати умову switch, тож про всяк випадок. Ханна: Але дійсно нічого поганого в цьому немає. Ми не очікуємо від вас демонстрації якихось шалених речей. Просто якщо інше - це цілком нормально. Це просто штуки, про які корисно знати.  І ось останнє, що треба сказати про категорію умови. У нас є умова в загальному вигляді, а потім маленький шматок коду, який працює, якщо умова істинна. А потім двокрапка, маленький шматок коду, який виконується, якщо умова хибна, крапка з комою. Ми закінчили з цим рядком. Синтаксично виглядає трохи потворно. Ми зараз подивимося на прикладі, я гадаю, що так буде зрозуміліше. Ми хочемо призначити рядку «професор «одне з цих двох значень, Девід Малан чи не Девід Малан, так? Ви маєте бути знайомі з рядком професор=. Ми надамо значення змінній, яка називається професор.  Тепер ми хочемо перевірити конкретну умову. У цьому випадку наша умова- num = = 50. Варто підкреслити, що коли у нас є рядок професор=, там потрібен один знак рівності. Це присвоєння. Тоді як у class\_num = = 50, два знаки рівності. Це перевірка якісної характеристики. Ми збираємось визначити чи номер класу дорівнює 50? Якщо так, професор - Девід Малан. Якщо ні, професор - не Девід Малан. Є питання? Знову ж таки це дещо, що варто знати. Ви можете робити так з якщо інше. Ви можете попрактикуватися вдома саме з цією умовою у іншій формі, тому що ви можете це зробити. Є якісь питання тут? Добре, я гадаю перейдемо до циклів. Чудово.  Шерон: добре, давайте поговоримо про зациклювання. По-перше, тут зліва ви бачите -- добре, це умова "поки". А потім ви робите ось так. Можна це змінити?  Джейсон Хіршхорн: Це вимкнення.  Шерон: Добре, і ще є фігурні дужки. Тож уявіть, що фігурні дужки знаходяться зверху і знизу, і зробіть це знову і знову. Якщо умова у круглих дужках оцінюється як істина, далі ви маєте робити те, що в циклі while .  Наприклад, ми можемо зробити практичне завдання. На мені зараз дві сережки, тож скажімо, якщо на мені хоча б одна сережка, поплескайте в долоні. Якщо ми стикаємося з цим, ми  [ плескають]  Так? І я одягаю одну, і --добре. Так. На мені більше немає сережок, тож більше не треба. Добре, ось тут ви маєте зупинитися. І це ви можете сказати дорівнює всім змінним. Скажімо, у вас є змінна кількість сережок. Коли число сережок більше або дорівнює 1, плескайте в долоні. І поплескавши, зменшуйте кількість сережок. Сережки мінус 1. Зменшуйте кількість сережок і ви пройдете через цикл двічі.  А якщо умова завжди істинна, скажімо якщо 2 = = 2, і так у всіх випадках, так? Тоді ви весь час будете робити дещо у рамках програми, і це еквівалентно майже вічному циклу, який був у Скретч. Так?  Студент: Дві сторони - це просто два шляхи написання циклу while?  Шерон: Ми сфокусувалися на першій. Давайте порівняємо його з циклом do while. Вони досить різні. Скажімо, умова досі коли я одягаю одну сережку, на мені зараз немає сережок. І досі треба плескати, коли я надягаю хоча б одну сережку. Що має статися?  [ оплески]  Ммм, панове. Добре, ви маєте плескати один раз, тому що ви маєте справу з першою частиною коду. Ви робите це , незважаючи ні на що, а потім ви бачите умову while, і знову повторюєте цикл, якщо умова істинна. Це мє сенс?  Студент: ви кожного разу робите це вперше. Шерон: ви завжди робите це вперше, незважаючи на істинність чи хибність умови. Джейсон Хіршхорн: як гадаєте, коли ви можете це використовувати? Коли це має сенс? Ханна: Так? Студент: [не чутно]. Джейсон Хіршхорн: вірно. Ханна: дуже добре. Джейсон Хіршхорн: коли ви даєте підказку користувачу, ви завжди це робите один раз. Тож замість циклу while, ви створюєте цикл do while, тому що ви завжди будете робити це один раз. Якщо вони видають вам правильний результат, завдання виконано. Якщо ні, ви попереджаєте їх знову.  Шерон: добре, що стосується циклів. У Скретч були блоки, що повторюються. Наприклад, ми хотіли повторити дещо, скажімо, 7 разів. І ми просто казали» повторити 7», і «я тут, щоб допомогти тобі, Білосніжка». У С, у нас є цикли, коли ми хочемо повторити щось певну кількість разів. Скажімо, якщо ми ініціалізуємо змінну «гноми» і впевнимося -- тож перший блок перед першою крапкою з комою, ми ініціалізуємо нашу змінну і присвоюємо їй значення 0. І наша змінна це ціле число, int. А ім’я змінної – «гноми», і ми встановлюємо їх значення - 0.  А друга частина між двома крапками з комами - це наша умова. І допоки гномів менше 7, ми будемо рухатися по циклу. І остання частина - що ми зробимо в кінці цього циклу? Гноми++ - і це означає, що ми прирощуємо гномів на 1 кожного разу.  Тож що тут відбувається? Спочатку ми повторюємо. У нас є значення гномів 0, і ми набираємо "Я тут, щоб допомогти тобі, Білосніжка!"! А потім кількість гномів зростає, тому що ми сказали гноми ++. Ми отримуємо одного гнома. А потім ми порівнюємо цього одного гнома. Ми маємо порівняти з умовою - гномів менше 7? Так. Я знову повторюю. "Я тут, щоб допомогти тобі, Білосніжка!" А потім гномів стає 2, і ми порівнюємо їх. Менше 7? Так. Ми повторюємо знов. І ми повторюємо цей цикл 7 разів.  Тож у кінці у нас будуть гноми=0, текст "Я тут, щоб допомогти тобі , Білосніжка!" кількість гномів дорівнює 1,2,3,4,5, і 6. Пам’ятайте, що ми рахуємо 0 теж. Ми починаємо з 0. Так?  Студент: тож це відрізняється від do, тому що не видає текст на початку. Ви можете продовжувати робити те ж саме. Ви не можете робити так само з циклом do? Цей же процес. Умова у тому, що гномів менше 7 чи щось таке. Відповідь - менше 7. Шерон: технічно ви можете. Якщо повертаємось -- ви говорили про цикл while, так? Цикл do while відрізняється тим, що ми гарантуємо, що ми хоча б раз пройдемо крізь нього, у цьому полягає найбільша різниця. Але коли є цикл while, ми можемо сказати поки гномів менше 7, робіть це, і потім збільшуйте кількість гномів на 1. А потім ми маємо ініціалізувати гномів, поки умова не вкаже, що вони дорівнюють 0. Тож так, ми можемо зробити так. Угу?  Студент: Можна повернутися до [не чутно] Шерон: у циклі? Студент: так, з гномами ++ виглядає так ніби це те, що ви робите після перевірки інших значень. Шерон: Ви можете не мати цього, а просто вкладати у -- Шерон: Так. Студент: Тож [не чутно] йде слідом за текстом у наступному рядку знизу. Шерон: так, можно помістити його туди. Але потім ви залишите його пустим.    Джейсон Хіршхорн: Проте вам все ще потрібна крапка з комою    Шерон: Це виглядає трохи дивно, але технічно ви можете це зробити. Технічно. Але не робіть так. Прошу.    Студент: є якісь альтернативи ++? Є щось інше ніж [не чутно]?    Шерон: Технічно це гноми. Одним знаком рівності ми встановлюємо гномів на значення плюс 1. Технічно це означає гноми ++. Зрозуміло?    Студент: так, але чи є якісь альтернативи? Типу як--  Шерон: так , ви можете зробити гномів--  Студент: [не чутно].    Шерон: Так, ви можете робити безліч речей. Ханна: ви можете збільшувати на 2. Ви можете збільшувати на 3. Все, що зміниться, буде підходити під вашу умову.    Студент: якщо ви хочете збільшувати на 2, як це написати?    Ханна: Ви можете написати гноми -- ви можете написати всю цю штуку. гноми=гноми +2. Або набагато простіше. Я напишу просто +=2.    Студент: Добре, а ви пишете це там де гноми++. Ханна: саме так. Шерон: вірно. Були ще питання? Так?    Студент: [не чутно] гноми++? Шерон: ви не-- Джейсон Хіршхорн: так, воно там не потрібно. [не чутно] у самому кінці. Тож ви робите [не чутно] одні, тому що ви типу дробите ініціалізацію, умову і зміну. У кінці вона не потрібна. Шерон: Також зверніть увагу, що вам не потрібна крапка з комою після всього циклу.    Студент: Як почати, якщо перше число від’ємне, наприклад?    Шерон: ви можете приписати гномам значення мінус 2.  Студент: ви просто ставите риску і 2?    Шерон: так, мінус риска 2.    Студент: чи це [не чутно] приписати гномам значення [не чутно]?    Шерон: так, якщо ми ініціалізували гномів раніше, просто пишіть int гноми крапка з комою, і тоді ми зможемо приписати гномам значення 0.    Студент: ми можемо зробити це раніше у програмі, сказавши int гноми= 0 . і потім просто --    Шерон: без-- там може біти просто відступ, але , можливо, все ще потрібна буде крапка з комою. Так. Угу? Студент: Цей код каже яким буде значення гномів опісля. Якщо ви спитаєте остаточну кількість гномів. Шерон: Ви можете попросити програму зробити це. Студент: Зрозуміло. Шерон: Але вам доведеться попросити. Ця функція не за замовчуванням. Я боюся знову надягти мої сережки. Зроблю це під оплески в кінці. Жартую.  Джейсон Хіршхорн: Це те ж саме. Приписуйте значення на початку. Умова , вивід, зміна. Це те ж саме, як якби хтось це бачив. Студент: чому немає крапки з комою у першому рядку? Шерон: де? Студент: Після першого рядка. Шерон: після 4 у круглих дужках? Тому що ми хочемо повторювати цей цикл. А якщо ви подивитесь на цикли у загальному вигляді, після них не стоїть крапка з комою. Це не практично. Зрозуміло? Ще одне питання, ще два. Так? Студент: Можливо, я забігаю наперед, але чи гноми відрізняються [не чутно] від змінних [не чутно]. Шерон: дуже добре. Студент: [не чутно]  Шерон: так, відрізняються. Що? Джейсон Хіршхорн: Так. Зазвичай область видимості змінної знаходиться у фігурних дужках. Тож область видимості завжди буде в фігурних дужках, але це не буде багато значити для циклу, тому що ми інціалізували гномів в цих фігурних дужках. Тож це звичайна область видимості гномів, на яку не поширюються значення змінних. Тож ви ініціалізуєте це у круглих дужках, а потім отримуєте область видимості. Це особливий випадок. Цього не має бути , і це було-- Зрозуміло?  Студент: ні.  Джейсон Хіршхорн: я повторю. Що вас бентежить у цьому?  Студент: ви можете використовувати гномів всередині -- ви очевидно можете використовувати гномів [не чутно], виходячи за рамки формули також?  Джейсон Хіршхорн: Ні, ні. Область видимості цього знаходиться всередині фігурних дужок для цього циклу, так. Шерон: Але ви ініціалізуєте гномів не в циклі, а потім ви можете використовувати їх у будь-якому місці. Є ще питання? ні? Добре.  Ми говорили трохи про вкладення циклів чи умови вкладення, якщо це інструкції. Ось приклад того, де ми можемо створити цикл. Скажімо, ми написали цю табличку іксів тут. Ми можемо хотіти -- перш за все, якщо ми не дивимося на код, а просто думаємо про нього, ми хочемо пройти через кожний ряд і написати кожну колонку, вірно? Зрозуміло? добре.  Зараз пройдемося по кожному рядку, а тут три рядки. І в кожному рядку для кожної колонки, а там є чотири колонки, print та х. Таким чином. Коли рядок 0 і колонка 0, ми виводимо на екран х. А потім ми йдемо через колонковий цикл. Рядок досі 0, але колонка 1. А потім 2, а потім 3. А потім ми виходимо з цього цикла тому що колонки вже не менше 4. А потім ми набираємо новий рядок, і переходимо до нового рядка. А потім ми йдемо крізь наступний рядок, і рядок збільшується, і ми проходимо крізь нього знову. Зрозуміло? Так?  Студент: тож щоб вкласти [не чутно] треба просто помістити всередину циклу?  Шерон: вкладення  означає, що у нас є циклз параметром усередині циклу з параметром, як у Початку.  Студент: вам не потрібен спеціальний поворот чи щось таке? Ви просто розміщуєте його всередині?  Шерон: Вірно. Прошу.  Студент: Можливо, це [не чутно], але виглядає так, ніби там зайвий простір між всіма іксами. Я не знаю чи це --якщо це має робити програма, це те, що вона робить?  Шерон: ні. Ми натискали Enter. Це наш недогляд. Перепрошую.  Х  анна: як ви зміните цю програму, якщо ви написали один зайвий рядок?  Студент: [не чутно]. Шерон: Молодець. Студент: Ви можете набрати те ж саме просто перемкнувши колонку, вірно?  Шерон: хм?  Студент: щоб набрати ваші маленькі ххх, перший рядок може сказати print, а замість рядка, він показуватиме колонку.    Шерон: тож ви можете змінити назви змінних, щоб зробити це. Ви це маєте на увазі?    Студент: ні , я маю на увазі що -- ви спочатку набираєте рядки, а потім колонки. Ви можете змінити порядок і отрмати ту ж саму діаграму? Шерон: так, але ви маєте бути обачні щодо того, де ви пишете новий рядок. І як ви повернетесь? Джейсон Хіршхорн: якщо ви просто переключили це і значення були відмінними від чотирьох -- що це означає? три на чотири. у вас буде чотири рядки і три колонки. Зрозуміло?    Студент: так.  Джейсон Хіршхорн: так, ви можете зробити це. Так, звичайно. Шерон: але це буде прсто зміна імен змінних. Добре? Так. Практичне завдання 1. У першій частині ви будете робити це, Маріо. Жартую. Більше схоже на це. І ми тільки що подивилися на вкладеннядля циклів, що створюють цю таблицю, подумайте, як ви можете набрати ці хештеги таким чином. А потім тут, як ви наберете це тут, цей відкритий простір.  Студент: [не чутно]. Шерон: так, ви просто набираєте простір. Добре, просто подумайте про це. Ханна: і у тій же частині практичного завдання програма під назвою greedy.c, щоб вам захотілося поміркувати про умови і впевнитися, що ви можете внести власні зміни. І одне маленьке попередження, яке ми маємо для вас: будьте обачні зі значеннями із плаваючою точкою. Якщо ви не розумієте про що я, про це буде розповідатися на наступній лекції на цьому тижні, і також [не чутно] у рев’ю, яке ви полюбите під час вирішення практичних завдань.  Річ , яку я настійливо рекомендую, особливо з Mario.c, коли ви виконуєте практичне завдання, якщо ви застрягли, починайте робити його не папері. Напишіть його і прикиньтеся комп’ютером, і перечитуйте - типу якби я був комп’ютером, як би я пройшов цей цикл? Як змінилися б мої змінні у циклі з параметром? Коли ви виконуєте завдання на папері - це у 10 разів легше, ніж за комп’ютером. Просто мій маленький лайфхак.  Шерон: і не думайте, що ви маєте кодити все зразу. Переконайтесь. що ви робите багатократний процес. Напишіть трохи, подивіться, як воно працює, подивіться, що відбувається. Це проба і іноді там бувають помилки. І відвідуйте консультації. Це дуже весело. Джейсон Хіршхорн: Є питання? Добре, панове, це було ваше перше заняття. Ханна: Дякую, що прийшли. Джейсон Хіршхорн: і я дякую. Шерон: спасибі . (оплески) |