Hello and welcome to Chapter 2. Now we're gonna just continue to talk about the building blocks of Python, variables, constants, statements, expressions, et cetera. The first thing we have to talk about is constants. These are just things we call 'em constants 'cause they don't change. There are numbers, strings, et cetera, and we use them to sorta start calculations or, you know, if something is greater than 40 hours we're gonna do something, so 40 is the constant in that situation. So, we have 123, we have 98.6, we have 'Hello World', which is a string by enclosing it in quotes, we pass each of these things to the print function, and a side effect of the print function is that we see the output. So print(123), prints out 123, print(98.6), prints it out. So these are just really the syntax of constants and without constants we can't write really much of anything. The other sort of foundational notion of any programming language are the reserved words, and like I said before, reserved words are these special words where Python is listening for them and there is very special meanings, so when Python sees if, it's not just any other word, it means how Python implements conditional execution. Variables are the third building block, and that is a way that you can ask Python to allocate a piece of memory and then give it a name, and you can put stuff in that. Sometimes you just put one value. Later we'll see, we do collections in chapters eight and nine. We will see that more than one value can be put into a variable. And the variable, how we control the variable, is through the assignment statement, and as I said before, it's important to think of the assignment statement as having an arrow to it. So this is not saying x for all time is the same as 12.2, what it's saying is take 12.2, find a place, find some memory in your computer there Mr. Python, give it a label x, we get to choose the x, that's the variable part, we chose it, right, and then stick 12 in it, and then the same is true for 14, go find another spot, name it y, and then put a 14 in there. So, think of this as an arrow every time you see that equality, the assignment, in an assignment statement. Now, these variables hold one value. So now if we have these three statements, these two, and then the third one executes, it says put 100 into x, but that wipes out the old value of 12.2 and it rewrites it with 100, and so we can change the variables; that's another reason that we call them a variable. There are some names, now insert some rules for making variable names. You can start with a letter or an underscore. We tend not to as normal programmers use underscore, we tend to reserve those for variables that we use to communicate with Python itself, so when we're making up a variable, we tend not to use underscores as a first character. You can have letters and numbers and underscores after the first character and they're case sensitive but it's really a bad idea to use case as the only differentiator. So, in this case, spam, eggs, spam23, and \_speed, are all totally legit. We would probably not use this one unless we were actually doing it because Python told us to use that variable. 23spam starts with a number, #sign is starts and dot is not a legitimate variable character. And spam, Spam, and SPAM are different, but this is not something that you want to sort of depend on too much, so that's just the rule names. We tend to start them with a letter and then use letters, numbers, and underscores. Underscores other than the first character are generally pretty common and you'll see those used a lot. Now, when we're choosing variable names, one of things about variables, is we get to choose the name; we get to choose the name x, choose the name y, And so sometimes we like 'em short but sometimes we want them descriptive, and the notion that of making variables descriptive is often confusing to beginning students. Sometimes it's really helpful to, if you're gonna have a line of text and you name the variable line, that's great, because the next person reading your program says, oh that must be the line of text. Whereas, it also can become misleading, that line, the name of a variable, somehow has meaning. So sometimes we'll have even singular variables and plural variables, like friend and friends, you know like is plural? Does Python know about singular and plural? And the answer is no. So sometimes we pick variables that make no sense, sometimes we pick variables that make a lot of sense. This is just something that you as a beginning programmer are going to have to understand that we can pick anything we want. And so, you'll see I'll try to call attention to this in the first few lectures as we go through. So here's a bit a code with an assignment statement, two assignment statements, a multiplication, and a print statement, and you can say what is this doing? Now, Python is perfectly happy with this code 'cause it assigns it in there, you have said please go give me this as a label, and then we assign two variables and then we're carefully pulling these two variables back out, multiplying them together, and sticking them into yet another variable and then printing that variable out. That seems like, you know, how we can figure out what it is, you just have to look really careful, and a single character mistake, and Python is gonna be, you know, pretty unhappy, okay? So that's one way to write this program. It's hard though, because any of those characters are long variable and they're random stuff, it's not very friendly to anyone who might read your program. Now this looks a little friendlier. It's the same program, because Python just wants a correspondence. You pick a, you pick b, and you pick c, and it's really much easier for us to see what's going on, and so this is, in a way going from here to here, is much friendlier, but, we can being even friendlier if we pick mnemonic variable names. So this is, this is not mnemonic, this is short and convenient, this is long and inconvenient. Python is happy with any of these. Here on the other hand, is another version of the exact same program, and now you think to yourself, oh yeah, now I get it. 35 is the number of hours, 12.50 is the rate, and then we're gonna multiply the hours and the rate and come up with the pay, and we're printing out the pay. Now whoever wrote this program is helping us greatly understand what's going on, and that's good. Choosing variable names, Python again, all three of these are the same to Python. Choosing variable names in a way that help your reader understand what's going on, is a great thing. The problem is, the danger is, if you read this and you think that somehow Python understands payroll; that if you name a variable hours that Python knows what hours means. The answer is, Python, really doesn't care what you name the variable, as long as what you name it you use it, right, and so you gotta be careful. And so you'll see, when I write my code in these first few weeks, first few lectures, I will sometimes write it with gibberish, I'll sometimes write it with extremely short but meaningless variable names and sometimes I'll use meaningful variable names and I'll call your attention to it and it will get you. You'll start, when you look at this third kind it has meaningful variables or mnemonic variable names, you'll just instinctively want to give Python more intelligence than it sort of deserves. I guess that's probably the best way to say that. So, we've talked about constants, we've talked about reserved words, we talked about variables. And so, here we have a sentence, like we've already done some of these things, where we said x = 2, we retrieve the old value of x and add 2 to it, so that becomes 4, and then we print 4 out. Print is a function that's built-in and we pass in whatever we want to print out, so this parenthesis is part of a function call. Okay, so an assignment statement, you have to really get your head around the notion that it has this arrow nature, and that it valuates this entire right-hand side before we change the left-hand side, and so you can think of this sort of as at time step 1 it does this and then at time step 2 it does the copy, and that's how you can have something like x on both sides of an assignment statement. And so, if for example we have x, and x has 0.6 in it, x has 0.6 in it, what happens is, is that, it first it sort of ignores this part right here and evaluates the expression. So it pulls the 0.6, everywhere x appears, it pulls 0.6 out, then it starts running these calculations, and then it has the new value. After all the calculations are done, then and only then is it going to put that back into x. And so it sort of takes that and puts it back into x, and then wipes out the old value. At this point, this has all been taken care and it's been reduced down to this 0.93, and so that is what's put in as the new value. So, up next, we'll talk a little bit more about making more complex expressions.