Week4 Lecture2.1

Hello, and welcome back to Chapter 2.

In this we're going to dive a little more in depth on the atoms,

the nuggets, the little pieces that make up Python.

And then we're going to work though our first real program that

has a sort of beginning, middle, and end.

So, here we go.

A big part of any programming language is the syntax for constants.

The nice thing about this it's kind of instinctive.

So, like a 123 or 98.6, numbers both integer and floating point numbers.

We've been using this on calculators.

They make a lot of sense to us.

The kind of constants that are a little bit different are things like string

constants.

So "Hello world" is a string constant.

We use that so that our program can say nice things to people that are using our program.

And so, we call them constants because they don't change but

it's kind of obvious, constants.

It's just a piece of the language, so that's a constant.

What else?

Variables and reserved words.

So the other thing other than constants are reserved words.

And as I said in a earlier lecture, these are like dog language.

Blah, blah, blah, blah, class.

Blah, blah, blah, del.

Blah, blah, blah, else.

Blah, blah, blah, lambda.

So these are words that when you say them to Python,

Python expects them to mean exactly what Python wants you to mean.

Now, why is this special?

It's actually kind of instinctive. It just means don't use these for other things.

Don't name functions, don't name classes, don't name variables.

Places where you get choices to use words.

Don't use these words. And most of them aren't too big of a problem.

I don't know, maybe break or in, or from, maybe.

Some of these you might naturally use.

Most of them are kind of weird enough that you're probably not going to use them as variable names.

And so the third atom basically is variables.

And you get to pick the variable names.

And it's a little weird that you get to pick variable names, but you'll get used to it.

And so variable names are places where you're asking Python to allocate a bit of memory and stick something in it and you're choosing the label.

And so if we take a look at this.

When we have this assignment statement, and remember that assignment statements always have a direction, right?

Think of these equal signs as having an arrow on them.

In some languages, I saw a language that uses kind of an arrow that uses a less than and a dash as the assignment.

I went, that's how it should be because equal confuses us.

Because equal means something different in mathematics than it does in Python or other programming languages.

So look at this very first statement.

What we're really saying is, it's a pretty complex statement.

After a while, you'll just use it.

Python, find us a spare piece of memory somewhere and give it a label of x, and put 12.2 in it.

And remember that, remember all that stuff.

That's one of things that Python does for us, it remembers stuff.

Then we go to the second statement.

It says oh, hey Python, find some spare memory that you got laying around, label it y and put 14 into that, okay?

And so that's how these things work and you chose x and you chose y.

Now if we keep on going in this code, and now we have another line and we say oh, hey, x, wait a sec.

I already told you about x, x already exists so don't go out and grab any new memory, but stick 100 in that.

So 100 goes in and it wipes out the old value.

So, that's what happens.

When this sequence starts, this happens first, second, third.

And so the third thing is the last thing and

so x ends up with a residual value of 100 in it.

So that's kind of variables and assignment statements.

You have some naming rules.

You can start variable names with letters or underscores, although we avoid underscores because Python tends to use underscores for its own internal purposes.

And the rest of it can be letters, numbers, and underscores.

And it's case sensitive.

But we don't want you to depend on that.

So, spam all lowercase, Spam with one uppercase, and SPAM.

These all are different variable names but you're not doing anybody any favor if you think that's being clever.

We tend to sometimes use uppercase to signal things, which we'll talk about later, but we tend to use mostly lowercase.

Some applications use what's called camel case, which is a mix.

But you tend not to depend on the fact that these are unique because you really are misleading the person who's reading your code.

But we have some good ones, spam, eggs, spam 23, numbers are fine.

It's fine to start with a underscore.

It's bad to start with a number.

It is bad to start with a pound sign, although that turns into a comment, and

you can't use characters other than letters and numbers in the variable.

And so those are all good variable names.

Now we lead these things into sentences or lines.

And so these things.

This is just another sequence of code, and it shows how you can also use the variable on the right-hand side.

So put 2 in some memory and label it x.

Pull that 2 back out and then add 2 to it.

So this little thing here becomes 4.

And then stick 4 into x.

And then this is a print function.

And the way functions work is they are the name of a function,

followed by a parenthesis, followed by another parenthesis.

And then you can print something, what you want to print.

And so, this causes on some output 4 to come out.

So, print is the function that you call to cause output.

Now I emphasize that one of the key things about variable names is that you

get to name them.

And we have a technique called mnemonic. I think I pronounced that right.

And the idea is that when you choose a variable name, you should choose a variable name to be sensible and

Python doesn't care whether you choose mnemonic variable names or not.

And the name that you choose for

a variable does not communicate any additional information to Python, okay?

So when you're a beginning student, sometimes if you use variable names that are too good it's confusing.

So you'll notice as I write, especially in these first two chapters, some of my code

uses really dumb variable names and some of them uses really clever ones.

So I go back and forth to emphasize to you that the name of the variable as long it's

consistent within a program doesn't matter, and Python is perfectly happy.

So here is a little code.

It's four lines of code.

And start at the top, run this line and that says find a little piece of memory

and label it that thing and stick 35 in it.

Find a little piece of memory and label it that thing, stick 12 in it.

Pull that first thing out and pull the second thing out, multiply them together, and put them yet into a third variable.

If you look at these things really closely, they're all unique and distinct, and you do have to look at them very closely. Then you print the third thing out.

And Python loves this code.

Python thinks great.

I don't know why you picked those variables, I do not care.

All I care is that they're unique.

Python really has no opinion about these variables.

But if you hand this code to a friend and then you say,

what is this code doing there?

They're like, why did you pick such insane variables?

because I have to stare at it and like, the difference between like these

two things is just the letter z and the letter p.

So this one we're pulling the p guy out and this time we're pulling the z guy out, right?

And these two just have a difference of those letters, I think.

I mean, you could do it.

Python loves this stuff.

To Python this is perfectly understandable code.

To human beings it is not perfectly understandable code.

So like people aren't going to like you, but Python is going to be very, very happy.

I can't draw a happy snake, otherwise I would draw a happy snake.

So, this is tacky, bad, not friendly.

So, let's make it a little more friendly.

You could choose variable names.

To those, Python looks at these two things as like, those are the same.

They're totally the same, Python doesn't care.

But now we can look at it and go, I see. The first variables is a and

we're putting that a thing there and the second variables is b.

We're putting the b thing there. And then we're multiplying them and going to put them in a c thing and then we're printing the c thing out.

That's pretty nice.

So this is a little better, right?

At least we can parse it and read it and understand what the code is going to do

without having to check to see if these two characters are the same or different.

So that's better but it's not mnemonic, okay?

Mnemonic means that we choose a variable name that makes sense for

what we're using it for.

Again, the variable name and Python looks at all these three things as the same.

All Python cares is that this matches this, this matches that and

that matches this.

But, humans are much happier.

Humans are much happier.

Now, we understand mentally that the first variable 35 was the hours.

The second was the rate per hours and then we computed a pay and we printed it out.

So this is really understandable.

The problem is for beginning programmers, it sometimes is too understandable, right?

It's too understandable. Right?

Huh? Python knows about payroll?

Because if I name a variable hours,

shouldn't Python know that that means hours?

And if I name a variable pay does that mean that Python knows about this?

And the answer is, no.

Python treats all these three as equal.

Python doesn't look at the name of your variables and think, hey,

you just had a bug, right?

So what if, for example, I rewrote this code and I said hours times hours.

And it's like wouldn't Python say hey that doesn't make sense.

Why would you even multiply it in a payroll calculation?

Why would you multiply hours times hours?

And the answer is, Python will happily do it

because it's not looking at those words.

But a human being who's trying to understand what your code is or

trying to help you, when you're writing code debug your code, will be very thankful that you chose good names.

And now I can look at your code, and if I see you say pay equals hours times hours,

like, why did you multiply hours times hours?

Didn't you really mean to multiply hours times rate?

So mnemonic variables are only for humans.

We will later see things that will confuse you when I name one variable

friend and another one friends.

And you'll be like, so Python understands plurals?

And the answer is no, Python doesn't understand plurals, okay.

It's really helpful to read stuff, so that when something has more than one thing in

it, you might maybe use a variable that's got a plural in it.

So, okay, enough about that.

I hope I made my point.

These are all the same to Python, but

human beings certainly prefer the second or the third.

And so, but be careful.

And after Chapter 10, you'll start using mnemonic variables and

it won't be a problem.

But in the first few chapters, you have to worry about it.

So as I mentioned, assignment statements basically are not like mathematics.

Equal sign means equality.

Assignment statements means arrow.

And the key thing is you can almost think of this as like there's a little wall

there and it completely computes this expression.

This is an expression, on the right-hand side,

gets that down to a single variable and then writes it into the memory location.

And that's why it's possible to have the same variable on both sides

because this side happens first, ignoring this side.

And then once this side is done, then it actually puts it into that other side.

And so, if, for example, x before the statement started, had a 0.6 in it.

It like firewalls this part off, and it computes this part here by pulling the 0.6, the old value of x, doing the computation, running through the things, getting the new value.

And then after it's got that new value, then, and only then,

does it store the 0.936 into x and that overwrites the 0.6 that was in x, okay?

And so one side finishes, produces a result, and then it can overwrite.

It doesn't mean you have to put the variable on both sides.

But it's why you can put it.

So sometimes we'll say something like x = x + 1.

And that's our way of adding 1 to x.

And that's because whatever x is, maybe x is 6 or something, this becomes a 7.

And then the 7 goes into x and then it overwrites it to the 7.

We call this increment.

We'll see this soon, okay?

But that's why you can put the same thing.

Now in mathematics, x = x + 1 makes absolutely no sense one but

in programming, it is one of the more common things that we do.

Okay, so that gets us through variables and constants and

now we're going to talk a little bit more in detail about expressions.