Week4 lecture2.2

So welcome back.

Now, we're going to continue on the right-hand side of assignment statements

and talk a little bit more about some complexity about

what you can do with these expressions.

So, all programming languages have various operators where you

say x+1 or x-1 or something like that.

These operators historically come back to come from the kind of

characters that were on computers keyboards in the 1960s, literally.

These things called Teletypes came from the end of World War II, and

they had a certain set of characters.

And we were going kind of from a mathematical character set,

where multiplication is a big cross or a dot in the middle and

exponentiation is raising a little tiny number above it.

And we just couldn't represent those on these really rudimentary key punches or

rudimentary terminals.

So we had to map mathematical formulas and functions into what could be done.

And so those keyboards from 40 years ago, 50 years ago, they had a plus on it.

So plus is a plus, they had a minus on it.

But multiplication, which is normally cross, is star. And division, which

you can't put things on top of one another, so you just made a slash for division.

And raising to the power is a double star.

Now, the remainder operator is something that's not typically on calculators, but

it's really important for computers and we'll take a look at this.

So, the addition operator, pretty straightforward.

I've kind of been using that without even talking about it, the multiplication, and

the division.

So, this is 5,280 divided by 1,000, which gives me 5.28.

But the one that's probably the most interesting is this modulo, or remainder,

operation.

And so, the way the remainder operation works is,

in particular if this is an integer, is it does the division.

So it's almost like a division, j divided by 5, but

instead of giving the quotient, this is the quotient, it gives the remainder.

So, 5 divided 23 is 4 remainder 3.

The 3 is what you get back.

Now you might ask yourself, why is this useful?

So one of the ways to make this useful is to pick a large random number and

then use the modulo operator, the remainder operator, with 52.

And then you end up with a number between 0 and 51, and then you can pick a card.

So you can take a random number, but you modulo at 52 and

now you can have a random number that is of card.

Or if you want to roll a dice, you'd make a big, random number and

you'd take it modulo 6 and then it tells you what side of the dice.

So, like games and stuff.

There's other situations where you can do even and odd calculations.

Like is this an odd number an even number? Divide it by 2 and

see what the remainder is.

And so, this whole notion of the modulo operator,

it's really useful in some situations.

And so, that's why we obsess about it a little bit.

The power operator, this is like 4 times 4 times 4, which is 64.

So those are the numeric expressions.

Now these also have an order of evaluation, and

the way it works is in mathematics, there's order of evaluation.

There are some operators that are more powerful than other operators.

And you can always, if you're clever, just put parentheses in, and

most programmers always put parentheses in.

So if I was writing this line of code, and I want it to be friendly to you so that

you could read it more easily, I would simply put the parentheses in for you.

So I'd say 5 to the sixth power goes first.

Then this 4 divided by that goes next.

Then this 2 times 3 goes next, and

then we evaluate the rest of these things left to right.

So I just added the parentheses that are the same.

This is exactly the same as what it would be without the parentheses,

because this happens first.

This part here happens second, this happens third, and

then all this other stuff happens fourth.

So there's an order, but we're going to teach you what the order is,

if you weren't going to use parentheses.

So the rule is, parentheses override everything.

Exponentiation is the most powerful, multiplication and division and

remainder are equal.

They're the next most powerful. And addition and subtraction. And then when in doubt,

you just go left to right.

So if there's a bunch of additions, you just go left to right.

If there's an addition and subtractions in a bunch of stuff,

you go left to right, okay?

So this is a classic exam question on computer science homework that

you'd just say, okay, what does this evaluate to?

Now of course, if you have Python,

you just type it in and it tells you what the answer is.

So the way I used to do these, back in the day when I was doing these homeworks,

because I would write the expression down on a piece of paper.

And then I would look through it and

I would figure out what the first thing was and I'd be very careful about this.

I'm like, okay, exponentiation happens first.

So I'll do the 3 to the third power which is 8.

And then I'll rewrite the whole thing, 1 + 8 divided by 4 times 5.

And then I would forget about this one, and I'd look here, and go, okay,

what's the first thing here?

Well, 8 divided by 4 is the first thing there.

So, then I'd be like, 8 divided by 4, well, that gives us 2.

In this case, it gives us 2.0.

And then I'd have 1 + 2.0 times 5.

And I look, and that's the next one, so that actually ends up being

10.0, and that ends up being 11.0.

And that's why this prints out 11.0.

And so if there are no parentheses, you can figure out.

Another kind of exam question is to ask what the parentheses would be.

So the exponentiation happens first, the division happens second, and

then actually you put another parentheses, because that result is multiplied by 5.

And so that's another way to kind of answer the same question is say put

the parentheses in where they belong.

Like I said, parentheses first, power second, multiplication third, addition,

and then left to right.

So when you've got nothing but addition and subtraction, you go left to right.

When you've got nothing but multiplication and division, you go left to right.

Start with the leftmost one.

Sometimes it doesn't matter.

In addition and subtraction, it generally doesn't matter, but in multiplication and

division, it can matter, left to right.

Okay, so we've been talking about variables,

we've been talking about constants, and we've been talking about expressions.

But we also have constants that are integers, we have constants that

are floating point numbers, and we have constants that are strings.

And we can manipulate these.

And Python carefully tracks not only what the value in a variable is but

what kind of a value it is.

So is it a string, is it an integer, is it a flowing point?

And sometimes this makes a difference.

And here's a little example code.

We have this plus operator.

And the plus operator is looking at its two operands, 1 and 4.

It's like, oh, those are integers.

I know what to do with integers.

That means addition.

And so, it adds this to be 5 and then sticks it into ddd and out comes 5.

On the other hand, we can actually use the same plus operator concatenating strings

because the plus operator looks to its left,

looks to its right, says, whoa, those are strings.

And I know what you mean.

I think you mean to concatenate this.

And so we end up with, hello, the space is very carefully constructed right there

because the plus doesn't add a space.

So this ends up being hello space there, and that little space is the space here.

And so Python knows, in this example, these are all constants,

but it knows type of constants, and it knows the type of variables.

And it can do very different things. And it can blow up and be unhappy,

as we'll soon see, based on the type of things.

So here we have an example of something that's unhappy.

So I have now concatenated hello and there, and

then Python's perfectly happy to do that, and then I stick that in eee.

And now I say I want to add 1 to it, and it looks to its left, looks to its right,

this is a string and this is an integer.

And Python says, I don't know how to do that.

Python could know how to do it, but it's chosen now to know how to do that.

And so you get the dreaded traceback, and traceback is like syntax error.

It's not Python telling you that you're a bad programmer or that you're never

going to be a programmer or that you're completely unsuitable for a programmer.

What Python is saying is I, Python, am lost.

You told me to do something I don't know how to do.

You need to go remember what it is that I'm capable of doing as Python.

Please come back and fix it.

It also means that the program stops.

Meaning that if you're in a multi-line script and you're doing a bunch of stuff

and there's a traceback here, the code after that quits, okay?

And that's because you've hit this line that Python is lost and

Python is loathe to continue.

Now we'll see in a bit that you'll be able to force it to continue if you want.

But because you said something that Python didn't understand,

it just quits at that point.

And traceback, line 1 in this case, tells you where it is that this thing blew up.

And so this looks like nasty gibberish but after a while,

it won't take you long at all to just relax, look for the word traceback.

That means Python quit somewhere.

It tells you where.

And then you look a little bit farther and it tells you what's wrong.

TypeError, it's unhappy with the type, and it's still referring up to this line of code.

It says, I can't convert integer objects to string implicitly.

So it's like you have told it to do something combining a string and

an integer.

And Python is like, I am lost, but I'm giving you as much clue as to how

I got lost, what got me lost, where I got lost.

I got lost, where I got lost, and what caused me to get lost, and

Python is trying as hard as it can to clue you in to what's going wrong.

So, don't look at this as like bad.

Look at this as like, oh.

I have not quite communicated to Python the way I want to communicate.

There's a way to solve this, there's a way to do something here, but.

So if we take Python and we think about Python and say, hey, you're so picky and

you blow up if I do the least little mistake on types.

Help me out here.

And so it turns out that Python has a built-in function called type.

So again, it's type with parentheses, function calls have parentheses on them, and

we pass something in.

And we're like, hey Python, tell me what the type of the variable eee is and

Python prints out, oh, it's a string.

What is the type of the constant 'hello'? That's a string.

What is the type of the constant 1? That would be an integer.

And so Python keeps track of the value of variables and

constants, the value and the type of it.

And so we have to kind of manipulate this and be aware of this as we move forward.

And so there are several types of numbers that we've already been

implicitly playing with.

Like I said, you can give a variable to type or you can give a constant to type.

Variables and constants that don't have decimal places are integers,

what are called integers.

And variables that have decimal places are called floating points.

Even if this was a 98.0, it would still be a floating point.

As soon as you put the point there, then that means it's a floating point.

They are represented internally differently.

Floating point numbers have a greater range than integer numbers,

but they're not always as precise as integer numbers.

So floating point numbers have

more range and less precision.

So integers are perfect but there's a limit, like four billion or

four trillion or something, there's a limit.

Don't worry too much about that.

There is a choice that you make.

You tend to use floating point numbers for things like temperatures or speed.

Turns out that you don't use floating point for money.

You don't use it for money, even though in our first examples,

we'll be bad and we'll use it for money.

But, shh, don't tell anybody that I'm using floating point for money.

If you want, you can play with floating point and

start finding some things where money goes bad.

Okay, there's actually is a number of different movie plots that come to have to

do with computer programs that use floating point for money.

I think Office Space is one of them that I particularly like, but

there's other ones as well.

Why you don't use floating point for money,

We kind of got off track on that, but that's okay.

So we also have a set of built-in functions that can convert from

one type to another.

There's float, there's int.

And so float is a function and it again has parentheses.

We pass in 99.

So as this expression is being evaluated, it has to call float.

It passes in 99, and then what comes back is 99.0,

which is a floating point representation of the number 99.

99 and 99.0 are not the same.

This is a floating point number.

So when this addition happens, it produces 199.0,

so that is a floating point 199.0.

By calling float, you are force converting that to a floating point number.

If we look at, like i is 42, we ask what type it is, well,

that i is an integer, we can take a variable and

pass it into the float function and get back 42.0 and 42.0 comes back.

And we say what kind of thing is an f?

Well, f is a float. Right?

So you can see how we're manipulating the type of things and controlling it.

And any time we get a little confused, we use the type function and say, hey,

what's going on here?

Why don't I understand what's going on here?

Integer division is something that actually changed between Python 2 and

Python 3.

It's one of the bigger changes, the bigger non-upwards-compatible changes

between Python 2 and Python 3.

And so this, I'm just reviewing.

There are some things that were different in Python 2.

So 10 divided by 2, even those are both integers,

produces a floating point result.

This kind of doesn't make much difference if these are evenly divisible.

But if they're not evenly divisible, it makes a big difference.

So 9 over 2 is 4.5.

Think about if you'd put in 9 divided by 2 in a calculator,

you wouldn't expect it to say 4.

And in Python 2, it said 4, it actually truncated, it actually threw this bit away.

But in Python 3, it just automatically converts divisions to floating point and

so it works out quite nice, okay?

So 9 over 2 is 4.5.

In Python 2, this used to be 4.

And then 99 over 100 is 0.99, exactly what you would expect in Python 3.

In Python 2, that was a 0.

Why?

Because it truncated.

It didn't even round the numbers, it chopped them off.

So this was really quite a silly artifact of Python 2.

In Python 2, if you get stuck in Python 2, you'd just use floating point numbers.

And once either side of the division or any other operation is a floating point

number, then the calculation is done in floating point.

So, if it's floating point input on either side, then it's floating point output.

And this is what we used to have to do kind of in Python 2 is force things to be

floating if we were doing divisions.

It wouldn't hurt to do it in Python 3, but

Okay, string conversions.

So if we read data, which we're going to see in a second,

from the outside world it comes in as strings.

Whether we're reading it from a network or from a database,

we tend to get these things as strings.

And so what we're showing here is a string value that's 123,

but it's not really 123, it's three characters.

It might as well be A, B, C, so 1, 2, 3, are the first three characters.

So, we take this string constant, we put it in there and we say,

okay, what kind of thing is in there.

It's a string.

And if we try to add 1 to it as we saw before,

you add a string and an integer and Python gets really unhappy.

Can't convert an int object to string implicitly.

But what if we know that inside of this string are actually digits, and we want to

get an integer representation, or a floating point representation of that?

Well, in that case we can call the int function, or

the float function, passing in a string and getting back an integer.

So, this basically reads these characters, the 1, 2, 3 and

goes, that's 123 and gives us back 123.

And we say, what kind of thing is in there?

Well, now, in ival there's an int.

Now I'm being mnemonic here.

I'm remembering that this is an integer and

this is a string, but Python doesn't care what I name my variables.

Remember, Python never cares about what I name my variables.

So, if I start naming them conveniently, don't all of a sudden think that

everything that starts with the prefix i is an integer, and

everything that starts with the prefix s is a string.

It'll be a number of lectures before I stop reminding you of that.

So, ival ends up with an integer, the number 123.

And now we can indeed add 1 to it, because it's an integer and

Python's happy, and out comes a 124.

So, this works pretty well unless the string that in question has no digits.

So, if there's no digits, it's going to blow up, it's like, whoa.

Now, let's read it.

Traceback means, I quit. Where?

Line 1, it's always line 1 because we're in this interactive environment.

But if you were in a script, it would tell you what line it is.

And then it says, invalid literal for int() with base 10: 'x'.

And that's like, okay, it's not working very well.

And so, you know, it'll say, that's an invalid letter.

And, again, we'll find ways to cope with this in later lectures.

Okay? Now, how do we get data from the outside world?

So, this is the keyboard, eventually we'll talk to networks and databases and files.

But right now, we want to take the keys from the keyboard, and

get it into a variable.

So, we have another function, a special function, the input function.

And when Python comes running in here, it starts the input function, and

the parameter to the input function is what's called a prompt.

It prints out who are you, and then it waits, and

then we type into the keyboard, Chuck, and then we hit the Enter key.

And then it takes this string right here,

and then it puts it in this variable right here.

It is a string. Even if I typed in 1, 2, 3, 4, it would be the string 1, 2, 3, 4, not 1,234.

Okay? And so, that is the way.

And so, this program at this point pauses until we type, and

we hit the Enter key, and then it takes that line of input including spaces,

whatever I type, and puts it in that variable.

And then the program continues.

And in this case,

it just prints out welcome, comma, and then the contents of the variable nam.

And in this case, this space right here that you see between welcome and

Chuck, that is caused by this comma.

We've mostly seen print being printing one thing, but

you can have as many things as you want with commas in print, and

every comma adds a space, and so, it's kind of friendly.

If you wanted to you'd have to concatenate these things together

to eliminate that space, but most commonly when you're

printing out a list of things you probably want spaces between them.

Okay? So that's how we read input.

Up next, we're going to combine all these things, and

we're going to make our very first program that does actually something useful.

So, we'll see you in a bit.