Week6 Lecture4.1

Hello and welcome to Chapter Four.

Now, we're going to come back and

visit what I alluded to in the first chapter of the fourth pattern for code.

Sequential, conditional, iterations, and then store and reuse.

This is the store and reuse pattern, and the basic essence of the store and

reuse pattern is that we, as programmers, do not like repeating ourselves.

So, if you have like four lines of code, and you want to do the same thing later,

and you put it down here.

And then, you put it later. And what if you find something wrong with those four

lines of code? Then you've got to find all the places, and,

let's just say you put it 100 places in your program.

You've got to find all 100 places and fix the mistake.

So we say well, why don't we put that one place and give it a name, and

then use it in all the other places?

And that's exactly what store and reuse is.

It's the idea of don't repeat yourself, D-R-Y, don't repeat yourself.

从 :1:1 开始播放视频并学习脚本1:01

So, this is what we've got here, stored and reused steps.

And functions are those things that we're storing and reusing.

从 :1:10 开始播放视频并学习脚本1:10

So, let's take a look.

So the new keyword that we've got is def, d-e-f, which stands for

start the definition of a function, define the function.

def is the keyword, it ends in a colon,

just like lots of things that start an indented block.

That starts an indented block.

You get the name the thing that you're storing, I call it thing for now, and

there's some some optional parameters, which we'll see later,

inside parentheses.

Right now, we have no parameters in this function, and

there is an indented block, and then when you de-indent that defines the function.

All right, so this thing has been de-indented, but it also

从 :1:44 开始播放视频并学习脚本1:44

this t is de-indented and so that is the definition of the end of the function.

Now, it's key to understand that as this is parsed or

looked at by Python it actually doesn't execute.

There's two executable statements there, print and print.

But all it does is it remembers it.

So it sort of like stores it into a little area almost like a variable,

right, like a variable x = 2.

Well, there's some x out there and you put a 2 in it.

Well, there is this thing out there, t-h-i-n-g.

And there's two lines of code in it.

So it's like a variable except that it holds code.

Okay, so a function is kind of like a variable, except that it holds code.

And then most importantly, it doesn't execute these lines of code.

So, no output, there's no output here.

Nothing comes out from that first part.

Now, it has a side effect of extending Python, so there's this new thing,

"thing", that we can call.

So we call this calling or invoking.

So we've made one of these things, now let's call the thing.

So we say thing().

And that's the syntax to say go call the function.

Now we've been using functions, print is a function, right?

print() parameter, that's how print works.

Well, thing is a function,

thing(), but there just happens to be no parameter in this because,

this is a really simple function.

So, what happens is, def does nothing but remembers thing,

then says oh, let's go back and run that thing, run those two lines.

So out come Hello and Fun.

And then it comes back down to here, runs this print statement, so out comes Zip.

And then it says oh, run that code again.

So this is the don't repeat yourself part. Run this, print, print, do it, again,

and now the program's done.

So it comes back here.

So one of the things about functions is that Python, when it calls a function,

it kind of remembers where to come back to.

It's like, okay, go up here.

Now I'm done, where do I go back to?

Oh, I remembered to come back to here, so then do this.

Okay, here's my coming back place, run up again,

run, go back to the coming back place.

So it's like it sort of pauses this code, pause here, run this, and then resume there.

So it's like jump and then come back.

从 :3:58 开始播放视频并学习脚本3:58

So we've been using functions all along, things like print, input, type,

float, int. int and float do conversions,

type tells us what kind of a thing something is, print prints stuff out and

input reads stuff from the user.

And then later we'll talk about functions that we're going to make and then use.

As we extend Python by building functions,

we can think of them as like new reserves or function names.

And so the naming conventions for function names is the same as for variables and

you want to avoid things like reserved words and other things like that.

从 :4:31 开始播放视频并学习脚本4:31

So, function's some bit of reusable code,

we define a function using a def keyword, and then we call or invoke.

We mostly say call but I think of invoke as a little clearer.

Like invoke this function,

do it right, here with a function name, parentheses, and then optional arguments.

So here's a function that we haven't played with yet called the max function.

It's also another one of Python's built-in functions and

this is an example of an argument.

We're passing in a string and we're asking max finds the largest of something and

in this case it's going to scan through this string and find the largest letter.

And it decides that w is the largest letter.

Apparently lowercase letters are bigger than uppercase letters.

And then it does what is called a return and it gives us back

the thing that it's like, you asked me to find this and now I found a w.

Here you go and then that w gets assigned into big.

So if we do a min, which is a different function,

we pass it in the same Hello world, it looks for the smallest thing.

And for some reason, space is the smallest thing.

So this space is what gets sent back to tiny, and

there's a space right there and away we go.

从 :5:39 开始播放视频并学习脚本5:39

So, at some level you can think of this max function as

a chunk of code that's been built into Python before.

We are passing in an argument, which is a string.

There's some code inside here that runs, reads through the argument, comes in and

then it reads through and looks at that

stuff, reads some stuff, and then sends us back the answer,

which is w, which is this is called return.

从 :6:5 开始播放视频并学习脚本6:05

It returns us something.

And then that w, so you think it's working on this side of this assignment statement.

The w is the residual value, once the max function is executed.

And then that then is assigned into big, and so big has the letter w in it.

从 :6:24 开始播放视频并学习脚本6:24

So the kinds of things that we've been playing with so far are type conversions.

And so here it goes Python, it's looking at what to print.

It's like oh, I've got an expression here but wait a sec, I want to do this but

wait, oh, I've got to call a function.

So there's a little float sitting here, a little code for float.

We pass a 99 in and we get back 99.0 and

then that 99.0 divided by 100 gives us 0.99.

So it sort of pauses its calculation and then it goes and

runs this float code and then comes back.

Now we're going to make a variable i, 42, say what kind of thing is it,

it's an integer.

Now, what we're going to do is we're going to pass this integer into that same

float code, 42 into the same float code, and out's going to come 42.0 and that's

going to replace this in the function and that's going to be assigned into f.

And so surprise, surprise, we have 42 in f, we ask what kind of a thing it is.

And then, it's going to do this calculation.

Remember, it's going to do the multiplication first,

the division second, so it's looking at 2 times float.

Oh, hold on for a second.

Stop for a moment.

We now have to run float, which means we take this 3,

we pass it into the float code.

The float sends us back,

in this case, 3.0 which replaces this in the function as 3.0.

And then we end up with this calculation of 2 times 3.0 or 6.0 and

then on and on and on the rest of it finishes.

So you think of these function calls as suspending what we're doing,

just for a brief instant, we're going to suspend that and then wait,

and then the function will give us something back to replace that, okay?

String conversions as well.

从 :8:17 开始播放视频并学习脚本8:17

We have been using int and float to do things like read things in

because input is a function but it always gives us back a string.

And it gives us a string. We might use this as a number. So the string 1-2-3 is

not the same as 123. That is a string. And if we do something crazy like add 1 to it,

Boom, we get a traceback because we can't concatenate string and

integer just cause they're the wrong type.

But if we take this same 123, and here's a little code called the int function.

从 :8:52 开始播放视频并学习脚本8:52

We pass sval in, which is 1-2-3 the string and

we get back 123 the integer then that replaces this in the function and

then 123 integer is assigned into ival.

And we say we hey, what kind of thing is ival?

Well, it's an integer.

We can add 1 to it and get 124.

And we just came from a previous chapter,

if we happened to call an int and we pass hello bob,

从 :9:22 开始播放视频并学习脚本9:22

in, I should have used a different color.

从 :9:27 开始播放视频并学习脚本9:27

We pass hello bob in.

This code blows up.

[SOUND] And it says oh, traceback, and that blew up.

So even a function can blow up.

And you know what?

Our code blows up.

So the function blows up and away we go.

从 :9:47 开始播放视频并学习脚本9:47

So up next we're going to stop using built-in functions and

actually define and use a few functions of our own.