Working with Functions and Modules

##### Speaker: Curt Knowles, Instructor

##### [00:00:00.650]

Hello again, everyone. In our next video looking at the Python language, we are going to show you how to define and use functions and modules in Python. What is a Python function? A function is simply a unit of code that performs a task.

##### [00:00:21.730]

These are often called subroutines, methods, procedures in other languages, and they're called functions in Python, and they're useful because they provide a way to divide code into manageable chunks or tasks. Using functions makes it easier to maintain, test and debug your code and functions can be reused by multiple programs. This is a huge benefit when it comes to maintaining a code base. The example that I always give is consider computing systems in different departments at a College or University.

##### [00:01:00.920]

Different departments in the College may have different reasons to calculate a student's grade point average. The finance office may want to calculate it for purposes of evaluating scholarships and grants. The registration office may want to use it in terms of calculating graduating with honors or grade point average for report cards. But it's a common function to calculate a grade point average. So instead of having functions everywhere in all different kinds of applications that each create grade point averages, you can create one function in one central code repository and have several applications

##### [00:01:47.840]

call and use that one function. Imagine if a University used a ten point grading scale, and then all of a sudden they made a decision to start using a seven point grading scale. If they did that, then your grade point average function, if each application consisted of a different function that calculates grade on average, that would have to be changed in many different places. But if you have only one function that calculates the grade point average and many applications call and use that same function, then you only have to make that change in one place, so it reduces your cost overhead in maintaining your programming code base as well

##### [00:02:25.940]

to use functions. In Python, the syntax for defining a function, we use a keyword called "def" that stands for defined function. And what we do in Python when we want to create a function is we use the def keyword, and then we give the function a name and we can provide zero or more arguments to that function, that would be input or data that the function needs to do its processing. Then we put our colon at the end of our function definition statement, and then we indent any statements that are going to be a part of that function four spaces inside of the function definition statement.

##### [00:03:10.340]

Here's an example of using a function that we have called print\_welcome.

##### [00:03:15.690]

The print\_welcome function receives no arguments, and all that it does is it prints a message "Welcome to the Future Value Calculator", and then it prints a blank line. Now, if we want to run this function from within our main program, we have to call it to run it. So the statement that we would use in our main program to call and run the print\_welcome function is just simply a statement that contains the function name and the parameter list, which in this case is empty.

##### [00:03:56.080]

Now you can also have a function that has one argument. In this case, we have our print\_welcome function again, and in this version of it, we have decided to send in a message to be printed by the function. If we were to call this function, first we would need to create a message variable and determine what will that message variable hold? In this case, it will hold our welcome message, and then we would pass that variable in as the argument to the print\_welcome function. Now I want you to notice as we move forward in this lecture, some functions return values back to the calling code, and some do not, and it really depends on the function's purpose.

##### [00:04:45.570]

If the function is only printing a message as we see in this screen, then it would not need to return anything to the calling code. The calling code doesn't need anything back from the function in order to continue running. However, if we call a function to calculate a value, then we would want to return that value back to the main calling program to be able to do something else with that value. Here is an example of a function that does a calculation and returns a value back to the main calling program.

##### [00:05:22.870]

So in this case we have our function called calculate\_miles\_per\_gallon.

##### [00:05:27.700]

We pass in two parameters, miles\_driven and the gallons. We do a calculation that calculates the miles\_per\_gallon based on miles\_driven divided by the gallons, and then we round that to just one decimal place. We place that value into a variable called mpg, and then we return the value in the variable mpg back to our calling program. So the way we would call this function is we would set two variables, miles and gallons. We would pass those in as a call to the calculate\_miles\_per\_gallon function.

##### [00:06:05.800]

And then we would have to set that on the right hand side of an assignment statement, with the left hand side being the variable that will receive back the value that is returned. So we pass in miles, which is 500, and gallons, which is 14. Our function accepts those values in the miles\_driven and the gallons of variables. It does the calculation 500 divided by 14, and places that value into mpg and rounds that by one down to one decimal place, which would end up being 35.7 being placed in mpg.

##### [00:06:46.760]

That value is passed back into mpg in our main calling program and mpg in the main calling program now contains the value 35.7.

##### [00:06:59.760]

When you're programming with Python, you often use multiple functions in your Python code and the way that you usually would type those into your code editor is you would place your functions first. So in this case, we would have function\_1, function\_2, and so on. And we would place those first in our Python editor. And then after all the functions have been declared and defined and written, then we would place our main program that would call and use the function\_1, function\_2 functions. So when you run this program, the first thing it does is it skips over all the def statements, the interpreter, and it first interprets the statements in your main program, and it runs those in order.

##### [00:07:45.740]

And then as it sees calls to functions, it will go transfer control up to those functions and execute those functions. And when the function is complete, it would transfer control back to the next statement in your main calling program. So let's give you an example. First of all, on this slide is the function code for a function called calculate\_future\_value. You can see here we have our function definition.

##### [00:08:15.100]

We send in three variables, monthly\_investment, yearly\_interest, years. You don't really need to concern yourself with the actual calculations that are going on here. Let's just say we're calculating the future value of a loan investment here. Okay. So we do some calculations using our passed-in variables, and what we ultimately return with is a value called future\_value, and we returned that variable back to a calling program.

##### [00:08:46.610]

So in our Python editor, we would have this function first at the top, and then after it, we would have what we call our main program. Now I specify sometimes you can,just to keep things straight, you can specify all of your main code in a defined function called main(). But you don't have to do that. You can just have all of this code to the leftmost indent inside of your Python editor.

##### [00:09:15.890]

So this would be the main code that executes first. That's going to use our calculate\_future\_value function. And you can see here part of what the program does is it sets up a while loop and it's going to calculate every time the user wants to calculate a future value of a loan. It's going to go through the process of setting up the variables that the function needs, calling the function here and returning back the future\_value into the future\_value variable. And then that future\_value variable that is returned from the function gets printed out and messages back to the user.

##### [00:09:52.440]

Okay. And then it asks the user, do you want to continue and do another one? And if they do, it repeats back to the while loop. If not, then it will drop out of the wall loop at that point and print "Bye". Now a couple of things that you can put on the parameters of your function call. The first one is your parameters can have default values.

##### [00:10:19.760]

So the default value is what would be used for the parameter or argument if you actually pass nothing into that argument in your call in your main program. So here we have a version where we have our calculate\_future\_value function and we have the three arguments. But on the years argument, we put a default of 20 by setting years equal to 20 in the argument list of our function definition. Now, if I were calling this from my main program, I could call it one of two ways. I could call it giving it two values, in which case these two values would be put into the first two arguments monthly\_investment and yearly\_interest.

##### [00:11:09.620]

And now, since I'm not passing in the third value, it's going to use the default of 20 for the years. Or I could call calculate\_future\_value, sending in all three values, and when it gets to the third value instead of using the default 20, it will override that with the value 10 that I pass in and use that instead.

##### [00:11:36.370]

The other thing that I can do is I can use named arguments. So with named arguments, I can pass the values in in any order. I can call the function from my main calling program using any order for the arguments in my function call because I explicitly clarify which argument each value pertains to. So if I'm calling calculate\_future\_value with named arguments from my main calling program, I would again do like I always do in calling the function name. But here I would clarify each one of my argument values by preceding it with the argument name and the function definition that this value should match up to and get put into.

##### [00:12:31.560]

So here I declare years first, monthly\_investment second, and yearly\_interest third. Even though monthly\_investment is first, yearly\_interest is second, and years is third in my function definition, because I use the names here on my call to the function, it will place 10 into the years argument, 100 into the monthly\_investment argument and 8.5 into the yearly\_interest argument.

##### [00:13:04.890]

Okay, one other thing with using functions is also to be able to use modules. A module is just simply a predefined Python function. Modules exist as their own .py files inside a standard Python library and are available automatically to you as the programmer to use. So you don't write - you can write modules, but that goes a little bit beyond the scope of what we're going to do here.

##### [00:13:34.100]

But mainly I just want to introduce you the fact that there are modules that you can use that Python makes available. So there are hundreds, hundreds and hundreds of these built in Python modules, and I've just listed a reference site here that you could go to at your leisure to view how many Python module libraries there are actually available. The one we're going to look at here is the temperature module. To use a module in your code at the very top of your code, you would put an import statement in the name of the module.

##### [00:14:09.820]

So if I want to use functions that are in the temperature module, I would say "import temperature" at the top of my code. Now I can also see what functions exist in a module.

##### [00:14:21.980]

So a module is a collection of functions. So to see what functions are available to me to use in the temperature module, I would simply use a help statement and say "help" and then in parentheses the name of the module that I want to look at the definition for. So if I put these two statements in here and I said "help temperature", this is what would display on my Python screen. It would give me the name of the module, a description of the module, and then a list of the functions along with the parameter arguments that you have to use

##### [00:14:59.620]

if you call these functions and it tells you what the function accepts and then what it returns.

##### [00:15:07.260]

The temperature module consists of two functions, a to\_celsius which takes in a Fahrenheit temperature and converts it to Celsius and returns the Celsius equivalent or a to\_fahrenheit function which takes in the Celsius degree and does a calculation and converts that Celsius degree and returns it as a Fahrenheit degree. So if I wanted to use this, I'm sorry. If I actually went in and looked at the actual code in the temperature module, I would see something like this. I would see my two function definitions for to\_celsius and to\_fahrenheit.

##### [00:15:48.640]

Those are the conversion functions that return the opposite of the converted degree, and then it would have a main calling portion that, you know would, what this is used for is if you're testing this module for any reason, you could test it just by simply calling the functions directly in the Python editor.

##### [00:16:19.940]

So really, as a programmer using this function, all you're interested in are these definitions. You will be calling and using these two definitions up here. You would not - unless you're actually writing the temperature module, you would not have any need for the main calling program or this if name == "main" program down here, which is used as what we call a boilerplate if you're testing this function. Okay, so really, the only thing you have to be concerned with on this page are the two definitions. This is what you would use if he were to call those functions from your your Python program after you imported the temperature module.

##### [00:17:00.660]

So we looked at the syntax for importing the module, says the word "import" and the name of the module. Now there are two ways that you can do this. You can just import the temperature module just by its regular name. And then when you call a function in the temperature module, you have to proceed it with a reference to that module name and the dot operator.

##### [00:17:25.660]

So if you are placing, if you had a Fahrenheit degree in a variable called "f" and you called temperature.to\_celsius passing in "f" as your parameter, it would convert that to a Celsius degree and put that in the variable "c" and vice versa. Here it would take the Celsius value in the "c" variable, convert it, and put the Fahrenheit value in the variable "f". An alternate way of using the module to kind of as a shortcut or a nickname, if you will, is you can import a module as another name.

##### [00:18:03.150]

We call this a namespace. So I can import temperature as temp.

##### [00:18:08.270]

It's almost like a variable for the temperature module. And then instead of having to type out "temperature" for every call to a function in that module, I can use the namespace, so I can say temp.to\_celsius or temp.to\_fahrenheit. So, it's just an alternative way to call the module kind of a shortcut, so you don't have to type out a big long module name every time you use something in the module. Now we can see a Convert Temperatures program that shows us how to use the temperature module and to use the conversion functions.

##### [00:18:43.990]

The output of this program would be, as you see here where we present the user with a menu. They can choose one of two menu options for conversion. Depending on which one they choose, it will ask them to enter a degrees Fahrenheit or a degree Celsius, and then once they enter that and hit return, it will calculate that and print out the corresponding degree Celsius or Fahrenheit. Likewise.

##### [00:19:09.840]

And then it will ask them, "do you want to do another conversion?" And it will continue doing that until they say no, I don't want to do another one. So this is what the code would look like for this program. The first thing we're going to do, we know we have to use the temperature module to use the conversion functions. So we're going to import temperature as temp.

##### [00:19:36.340]

Next, I'm going to go down to my main code, and the first thing I want to do is display a menu. So I'm going to use a function to do this. I'm going to call the function display\_menu, and I'm going to call that function. In that function, I'm going to write the header menu and the two menu options and a blank line. Now, after I display my menu, I'm going to set up a loop control variable called "again". As long as "again" is equal to yes, I'm going to keep asking for another conversion.

##### [00:20:13.150]

Okay, so while "again" is "y" I'm going to call a convert\_temp function. In that convert\_temp function, I'm going to ask the user enter a menu option. If they enter 1, that means they want to convert from Fahrenheit to Celsius, and I will ask them to enter a degree Fahrenheit. When they do that, I will call the to\_celsius function of the temp module. Then I will round that to two places and I will print out that result.

##### [00:20:46.020]

If they enter 2, I'm going to ask them to enter degress Celsius, I'm going to convert it to Fahrenheit, round it to two places and print the result. Otherwise, they did not enter a value menu number if they didn't enter 1 or 2. So I'm going to ask them to enter a valid menu number and then that finishes my convert\_temp function call. So then I'm going to print a blank line and ask them, do you want to convert another temperature again and again?

##### [00:21:18.990]

If they say yes, I'm going to go right back up to the while loop and run convert\_temp again, asking for another menu option and do another conversion. And I'm going to keep doing this over and over again until they enter no, in which case I will drop out and print the message "Bye". Okay, so that finishes this introduction to functions. Another thing that I'm going to do in another video in this module is I will give you a Python demo of solving a problem that needs to use functions.

##### [00:21:54.580]

Okay. Thank you very much. Have a good evening. Bye.