Python Programming Basics

##### Speaker: Curt Knowles

##### [00:00:00.000]

Hi everyone in this video we're going to talk about some Python programming language basics. Some of this material will be review at this point, and some will be new. But we are establishing a good foundational basis for programming in Python here. First of all, a quick review of low level versus high level languages. So a low level language is a language that machines understand.

##### [00:00:36.040]

It involves a slight abstraction if any from machine code. It's usually written in binary and is fetched, decoded and executed by the CPU at a machine level. A high level language is a language that humans understand. Some examples of high level languages today would be C, C++, C#, Java and Python. Python is the language that we are programming in in this class.

##### [00:01:13.340]

It is a high level language, and it is an interpreted language, which means that when you run your Python program, it evaluates each statement in your program one at a time against the Python language syntax rules to guarantee or to ensure that the program can successfully run without error. Pycharm is an integrated development environment, also called IDE, that we can write Python code in. PyCharm is integrated because it provides several tools for us to be able to work with Python. It provides a text editor in which we can type our programs, a file organizer and explorer that helps us to organize all of the files in our projects, and an interpreter and debugger that allows us to run our finished programs directly in PyCharm, see the results in PyCharm and see any errors, and it makes a set of debugging tools available to us to help us to debug and fix any errors in our program.

##### [00:02:27.700]

Another thing that we must discuss before we actually get into looking at basic language syntax of Python is the entire concept of algorithms and the process of designing a program. No matter what language you write in, programs must be designed before they are actually written, and there's usually a development cycle that's involved in program development where you first design the program, then you write the code from the design, you run, debug, and correct syntax and other errors in your program. You test the program fully against a unique set of constraints and test scenarios, and you correct other errors in your program based on those scenarios, and you continue that cycle until the program is declared to be what's called QA or quality assurance approved, and at that time it can be deployed and moved to a production environment where live users are actually using it.

##### [00:03:36.800]

So design is the most important part of this cycle because it sets the foundation. You first have to understand the task that the program is to perform.

##### [00:03:48.920]

So usually what happens in the real world is that you work with a customer to get a sense of what the program needs to do for them. You ask them questions about program details, and based off of their answers, you create one or more software requirements. You determine the steps that have to be taken to perform the tasks that they need done. You break down each required task into a series of steps. Thus, you create an algorithm - a list of logical steps that must be taken to perform that task.

##### [00:04:25.290]

One of the steps in creating an algorithm is to write pseudocode or fake code. This is an outline in English language, kind of an informal language. It has no specific programming syntax rules that you have to follow around it. This is not a program that's meant to be compiled or executed. It's simply an outline.

##### [00:04:49.660]

It's used to create a model program from. There's no need to worry about syntax error, so you can focus on the program's design in your pseudocode, and it can be translated directly into actual code in any programming language. So let's take a real life problem. First, non-computer related, to show you that even in the simplest tasks that we perform in the real world involves a series of steps or what you could call an algorithm. So let's take the problem of having to boil a pot of water. Even though most of us know how to boil pot of water without even thinking about it, there are steps.

##### [00:05:33.130]

If you were to write them down, first of all, you have to go get a pot, then you have to pour a desired amount of water into the pot, and then you have to put the pot on a stove burner, and then you have to turn the burner to high, and then you have to watch the water until you see bubbles rapidly rising. And at that point you have successfully completed or resolved the problem of having to boil a pot of water. So if you think about it in that sense, it helps you to be able to see computer related problems in an easier light.

##### [00:06:08.980]

Let's take a pay calculation problem where we have been tasked with writing a program to calculate and display the gross pay for an hourly paid employee. So what are the steps here?

##### [00:06:24.760]

Well, first of all, we need some information. We have to do a calculation, and the calculation is to calculate gross pay for an hourly paid employee. So there's a couple of pieces of information that we need here to do that. First of all, we need to know the number of hours that the employee worked. Secondly, we need to know what is the employee's hourly pay rate.

##### [00:06:50.250]

After we have those pieces of information, now we can perform our calculation as a third step. We can multiply the number of hours worked by the hourly pay rate. And then finally, now that we have the result of our calculation, we can display that result back to the user in the output of our program. So let's take the problem of the pay calculation and let's convert it from algorithm to pseudocode. Pseudocode is just like the algorithm.

##### [00:07:25.910]

Same steps. You might add a few steps, but it's more of a "computerish" type language. You're starting to think more like the computer would look at this problem. So instead of getting a number of hours, we would put maybe two bullet points in our pseudocode.

##### [00:07:46.690]

One for, okay, we have to display - what are we going to display to the user, "Enter the number of hours the employee worked". That will be our prompt. Now we're going to let them, we got to wait on them to input the number of hours. And then after they input that, then we're going to display "Enter their hourly pay rate", and we'll wait for that input. Once we have that input, now we will calculate gross pay.

##### [00:08:12.820]

What is that calculation going to look like? Well, to calculate the gross pay, it's going to be the number of hours times the pay rate that will give us their gross pay. And you notice here I use interesting looking names. payRate, grossPay, hours - and payRate is not two words. It's one word with, you know, the first word "pay" being lowercase, the second word "Rate" being capitalized with the R.

##### [00:08:44.080]

So what I'm starting to do here is I'm starting to put what I call variables into my pseudocode. These will be actual variables I will probably use when I actually write the program to hold values for these pieces of information. And then finally, once I have the grossPay, what am I going to display? I'm going to display "The employee's gross pay is $" and then their gross pay. So typically, computer programs perform a three step process.

##### [00:09:17.020]

And we just saw that in our pay calculation example, we receive two pieces of input from the user, and then we perform some process or calculation on that input. And then finally, based on the results of the process, we produced output. So these three steps - input, process, output. We refer to them in programming as IPO. And when you are programming, you should always think of solving problems in terms of IPO.

##### [00:09:47.080]

What inputs do I need? What calculations do I have to perform on the inputs and what is going to be displayed back to the user?

##### [00:09:58.000]

Okay, so now we're ready to get into actual Python a little bit. So as we do that first, let us go over just a few very general coding rules to working with Python. First of all, Python relies on proper indentation. Standard indentation is four spaces. You'll see this more as we get into Python, especially when we get into working with control structures like if statements.

##### [00:10:24.380]

and while loops. You can write one statement on multiple lines in your text editor in Python, if you use the \ character at the end of each line of your statement, and normally statements will execute when you run the program in the order that they appear from top to bottom. Also, let's just mention comments. You can comment inside of your Python program. These are notes of explanation that are really just meant for you or any other programmers that may come along behind you that would work on the project at a later date.

##### [00:11:02.560]

These comments are statements in your text editor that begin with a # character. You begin these statements with a # character.

##### [00:11:11.940]

This # tells the Python interpreter when it runs the program. to ignore that statement. You can have an end-of-line or inline comment or a block comment. An inline comment might appear at the end of a line of code and would describe the purpose of that one line of code, whereas a block comment would appear on its own line and describe a whole section of code. So why is commenting important? Why do we mention it?

##### [00:11:38.460]

Well, in real life, it helps in a couple of ways. It helps you to debug errors in your own code. You can comment one line at a time until your program runs without any errors, and that'll tell you where the last statement commented was probably the one that committed the error. It also helps programmers after you to understand difficult portions of code logic and what you're originally trying to do when you wrote the code. Now we get into some of the language syntax of Python.

##### [00:12:09.040]

We look first at variables and data types. A variable in Python is a name that represents a value stored in the computer memory that is used by the program. An assignment statement is used to create a variable in Python and make it reference a data value. So the general format of an assignment statement is you have a variable on the left side of an = expression, and then you have your value or other expression that represents the value on the right hand side of the = statement.

##### [00:12:44.290]

"age = 29" is an example where we are taking the value 29 and assigning it to the variable "age".

##### [00:12:52.350]

You can only use a variable in Python if you have a value assigned to it. So if I tried to print "age" without first assigning it a value, then my Python program would error when I tried to run it. There are rules for good variable naming in Python. First of all, the variable name cannot be a Python keyword. Keywords are words that are reserved by the Python syntax language for specific purposes.

##### [00:13:25.920]

A variable name cannot contain spaces. The first character of the variable name must be a letter or an underscore. After the first character, you can use letters, digits, and underscores. And variable names are case sensitive, so that's a very important point that you need to be careful about, and it's one that often students kind of trip up on as well. Not really realizing that Python is a case sensitive language. Every time you choose a variable name,

##### [00:13:59.800]

it should be descriptive and reflect what is that variable being used for? So instead of using "x" or "y", you would want to use like payRate, grossPay, hours, hoursWorked. A good practice to keep consistent in your variable naming is to either separate them with underscores, as we see here with miles\_per\_gallon, or to use what's called camel-case, where the first word the variable name is a lowercase letter and then every other word is starts with an uppercase letter.

##### [00:14:41.260]

Data types define the type of data that you are using in a specific variable. There are three main Python data types we'll look at now. One is a string or an "str" in the Python syntax. This represents any character or text data and you can see some examples here.

##### [00:15:00.690]

We also have integers ("int") which are whole numbers. These can be positive or zero or negative, and we have "floats" which are floating point numbers. This is the set of numbers that require decimal points and you can see some examples here. These can also be negative. Variables can also be reassigned to new values during program execution.

##### [00:15:26.170]

When you do this, Python will remove the previous value that was assigned to the variable, and the new value being assigned to the variable can actually have a different data type than the previous value. This is one of the places where Python differs from almost every other language. In Python, one variable can represent different values of different data types throughout the life of the program, and that's not usually the case in other languages. Once you declare a variable in other languages, it has to be one specific data type throughout the life of the program.

##### [00:16:01.500]

A couple of things to note also about representing strings and numbers (integers and floats) in code.

##### [00:16:10.210]

You can represent them as string or numeric literals. If you represent a string literal, that would be an actual string in double quotes inside of your code, numbers would be just numbers inside the code with no double quotes around them. Also notice that numbers can also be expressed as string literals, in which case the string literal is different than the numeric literal. So "5" does not equal the number 5. It equals the "character 5".

##### [00:16:43.880]

So there is a character "5" as well as there being a number 5. So here is just a chart showing the overview of the three data types we've talked about and some examples of setting assignment values to variables for the three types. And over here in the comments, it tells you what data types are being assigned to each one of these variables. If we come down to the second set of statements, we see that here we are reassigning the variable first\_name from "Mike" to "Joel" so the value "Mike" would then go away and first\_name would contain the name "Joel".

##### [00:17:26.960]

Also quantity1 gets reassigned to the numeric literal 10, and then quantity1 gets reassigned again to another variable quantity2, which is the number 5.

##### [00:17:40.520]

So this puts the value of 5 into quantity1. Finally, we attempt to set quantity1 equal to the string literal "15", not numeric, and Python is able to do that even though it's a different data type. And now quantity1 is actually a string with the value "15" in it? The coding statement at the bottom would actually cause an error in Python.

##### [00:18:09.150]

And the reason for this is what we said earlier about case sensitivity. Quantity2 - we do have a quantity2, but it has a lower case "q". This Quantit2 begins with an uppercase "Q", so this is not the same variable as lowercase quantity2. So nowhere, if this was one contiguous program, nowhere in our program have we assigned a value to Quantity2 that starts with uppercase "Q".

##### [00:18:35.900]

So here we are trying to use Quantity2 without first assigning a value to it, so that will give you a "Name Error" when you run the program. Quantity2 is not defined. Here's a complete list of Python keywords. These are the words that you cannot use as variable names because they're used for other specific reasons by Python. And here's an example of the underscore character name variable name method and the camel-case variable name method that I mentioned before.

##### [00:19:11.260]

Now let's look at displaying output in Python using the print function. A couple of definitions. First, a function is a unit of prewritten code that performs a specific operation, and arguments are pieces of data that are given to a function that allow the function to perform its task. So one of the built-in functions in the Python language is the "print" function. The print function,

##### [00:19:41.320]

its purpose is to display output to the screen. Arguments that are sent to the print function are the pieces of data that are to be printed onto the screen. Python does allow multiple arguments to be passed to a single print statement. When that happens, arguments are separated by commas,

##### [00:20:06.160]

they're displayed in the order that they are passed to the print function, and they are automatically separated by a space when they are displayed on the screen. Here are some examples at the top. We have the official formal definition of the print function. We have three example print statements here that prints out "Hello out there!", the print with no arguments prints a blank space, and then print "Goodbye!", prints the word "Goodbye!", and notice how it prints that on three separate lines. So after each print statement, it does a carriage return line feed to print the next statement on the next line.

##### [00:20:50.440]

The print function - here are some examples of where the print function can receive one argument or multiple arguments, and the arguments can be a different type. So here we have a string literal "Price" and a float literal 19.99. Over here behind the comment is what you see would be output to the screen. Remember, we said the blank space would be between, would be printed between the two arguments. The arguments are separated by commas.

##### [00:21:21.880]

So here we pass numeric literals 1, 2, 3, and 4, and they're each printed each separated by space. So that's how you print output to the screen in Python. So how do you read input from a user? You do that with the "input" function. The input function has the format of calling input with one argument, and that would be your prompt.

##### [00:21:48.680]

That's what will appear to the user as the prompt on the screen. When you would usually set that on the right hand side of an assignment statement, and you would set a variable equal to that. So the value that would get passed through the variable would be the data that the user inputs. So if you say "Enter username:" and they enter the name "Curt", the name "Curt" would be put as a string into that variable. Here's the official definition of the input statement in the Python language.

##### [00:22:23.670]

And here are some examples. In the first example, we have an input statement that puts ask the user to enter their first name and put what they enter into the variable first\_name, and then finally we're on to print "Hello, " and their first name. Notice here that instead of separating multiple arguments with the comma, we can also separate them by putting the + sign, that's saying concatenate or join these together. And here would be your output where the user enters "Mike" and the output is "Hello, Mike!". Another way to get input from the user would be to have a print statement.

##### [00:23:10.600]

Just print out the prompt and then do your input statement on a separate line with no arguments. So this would again wait on the user to input their name and put in the first name, and you would print your "Hello first name" message. But this time because you put these two statements on two lines, print and input, it would print your prompt on one line and on the next line it would take in the input from the user. So that is how that differs an output from the first example.

##### [00:23:41.720]

Now here's another example where we set a score\_total, which will now be a numeric integer variable, to 0. We take in the input "Enter your score" from the user. We put it into the variable score, and now we do a mathematical operation. This += means "add the amount that was input as score to what the value of score\_total is". When we try to do that, however you run this, you're going to get an error.

##### [00:24:17.690]

And why is that? It's because every time you use an input statement, it doesn't matter if the user enters a string or a number, it will always come in as a string. And so it will set score equal to a string. So if I say "Enter your test score" and the user enters 90, that comes in. But score is not the integer 90, it's going to be the string "90".

##### [00:24:47.330]

So to get around this error, I have to do something to convert that string "90" to be the number 90, the integer 90, so I can then use it in math operations and calculations. So that's what I I have here is that it always comes in as a string. But you can use the two functions "int" and "float", passing in the item that was input by the user to those functions to convert that item either to an integer or to a float. So some examples of why integer? When to useinteger and when float?

##### [00:25:29.590]

Just think about "is this something that should include a decimal piece"? So if I ask a person's age, age is never going to be expressed in decimals. It's always a whole number. So age would be an integer. But if I ask for a price, that would include dollars and cents, so it would include decimal places, so price would always need to be a float. You can set this up with integer.

##### [00:25:54.480]

So now we're using two functions. We're using the input function and the int function. So you can set this up using what we call "chaining" or nested functions where you take in the input of in the prompt here and then the string that comes in, you use the int function as function1 to convert that. So the value returned by the input function2 is passed to the Int function1 to be converted.

##### [00:26:23.940]

Now here are some examples. First of all, comparing or adding, trying to use strings in math operations. We tried to use the string "5" in the addition - that's not going to work. It's not an integer. So to get around that, we put the int function around "y" to convert it to the number 5, and then we can get our numeric result. Here is an example where we get the input from the user, and then as soon as we bring it in as string quantity, we convert it to an integer quantity using the int function. We can also do this one statement does the same thing as these two statements where we input the data and then we immediately convert it to an integer and put the result into the quantity variable. Same thing with the float.

##### [00:27:17.710]

We can do it in two statements, or we can do it chaining as one statement - a nested function call.

##### [00:27:26.940]

Okay, so we've looked at variables, data types, input, output. Now let's talk about the process of performing the calculations. So of course we're going to use math expressions to perform calculations and return values, usually to variables. So the math expressions, as in mathematics, include an operator and two or more operands. You have a list of Python's arithmatic operators here - the basic ones, and these are the characters you have to use, the +, the -, the \* and the / to indicate addition,

##### [00:28:05.730]

subtraction, multiplication, division. You can also use a // to indicate integer division. So with regular division, the result will be just like a usual division, where if the result is a floating point number, it will be expressed as a floating point number. But with integer division, if the result is a floating point number, then the decimal portion of the result will be dropped and only the integer portion will be retained. Another couple of notes here - the modulo or remainder (%). This will do a division and will return the remainder of the division as the result.

##### [00:28:43.260]

Exponentiation will take the first operand to the power of the second operand. Over. Here we see examples of addition, the floating point division, the regular division that gives us a float, and then that same division using integer division which drops the decimal part and returns just the integer part. Now 25/4. The remainder of that would be a 1 and 3 to the power of 2, the result of that would be 9.

##### [00:29:14.290]

Now, just as in mathematics, you can pay attention - you need to pay attention to order of precedence when doing math operations. I don't know if you remember the old acronym from Math - PEMDAS: P-E-M-D-A-S. The same applies here. The tier and the levels at which precedence is evaluated is

##### [00:29:39.520]

first, it looks for the "P" - the parentheses, any parentheses gets a value first, anything inside parentheses. And then the "E" - exponentiation. Any exponentiation is done next, and then the "MD" - multiplication and division, whether it be regular division or integer division. Also modulus now will fall into this tier level of precedence, and then finally at the lowest level is "AS" - your addition and your subtraction. The higher precedence gets performed first.

##### [00:30:15.320]

But if we come to a statement that has multiplication and division on one line, or addition and subtraction, that will get evaluated from left to right through the operation. Now, depending on what data types you are performing math operations on, it will change - talking about numerics here - it will change the resulting data type, the data type of the result. So if you're performing an operation on two integer values, the result, of course, is going to be an integer. On two float values,

##### [00:30:49.680]

of course, the result will be a float. However, if you doing a math operation on one integer and one float, the integer will be temporarily converted to a float, and the Python interpreter will do this by adding a .0 to the end of the integer, and then it will perform the operation and will express the result of the operation as a float. This type of expression, because it contains multiple data types, is called a mixed type expression. Now, if you ever get in a situation where the type conversion of a float gets converted to an integer.

##### [00:31:32.770]

Then all that will do is it will truncate similar to the integer division.

##### [00:31:37.980]

It will truncate the fractional or the decimal part of the floating point number so that number can be, that result can be expressed as an integer. Some examples here, along with order of precedence with exponentiation, multiplication, divisions, and modulo, addition, subtraction, the effect of parentheses on the order of precedence. You can see here where the 4\*5 in the first one is operated on first, but the parentheses changes that order and operates the 3+4 first, and that changes your result accordingly. Also, here is four statements of code that calculate sales tax.

##### [00:32:27.180]

Another example, I have a float 200.00 and a tax\_percent float .05.

##### [00:32:34.680]

I'm going to multiply that two numbers to get the tax amount, and then I'm going to add the tax amount back to my subtotal of 200.00 to get a grand total, which will be 210.00. Here's code that calculates the perimeter of a rectangle. So I have a width and a length, and the mathematical formula for perimeter of a rectangle is two times the width plus two times the length. So using parentheses and also using my math operator and addition operators, I can perform this calculation 2 \* width is 8.5, 2\* length is 17. Add the numbers together and

##### [00:33:19.370]

your perimeter is 25.5. And that concludes this first lecture on Python introduction and the basics of the Python language. Thank you very much.