**[Lecture 0](https://cs50.harvard.edu/python/2022/notes/0/" \l "lecture-0)**

* [Creating Code with Python](https://cs50.harvard.edu/python/2022/notes/0/#creating-code-with-python)
* [Functions](https://cs50.harvard.edu/python/2022/notes/0/#functions)
* [Bugs](https://cs50.harvard.edu/python/2022/notes/0/#bugs)
* [Improving Your First Python Program](https://cs50.harvard.edu/python/2022/notes/0/#improving-your-first-python-program)
  + [Variables](https://cs50.harvard.edu/python/2022/notes/0/#variables)
  + [Comments](https://cs50.harvard.edu/python/2022/notes/0/#comments)
  + [Pseudocode](https://cs50.harvard.edu/python/2022/notes/0/#pseudocode)
* [Further Improving Your First Python Program](https://cs50.harvard.edu/python/2022/notes/0/#further-improving-your-first-python-program)
* [Strings and Paremeters](https://cs50.harvard.edu/python/2022/notes/0/#strings-and-paremeters)
  + [A small problem with quotation marks](https://cs50.harvard.edu/python/2022/notes/0/#a-small-problem-with-quotation-marks)
* [Formatting Strings](https://cs50.harvard.edu/python/2022/notes/0/#formatting-strings)
* [More on Strings](https://cs50.harvard.edu/python/2022/notes/0/#more-on-strings)
* [Integers or int](https://cs50.harvard.edu/python/2022/notes/0/#integers-or-int)
* [Readability Wins](https://cs50.harvard.edu/python/2022/notes/0/#readability-wins)
* [Float Basics](https://cs50.harvard.edu/python/2022/notes/0/#float-basics)
* [More on Floats](https://cs50.harvard.edu/python/2022/notes/0/#more-on-floats)
* [Def](https://cs50.harvard.edu/python/2022/notes/0/#def)
* [Returning Values](https://cs50.harvard.edu/python/2022/notes/0/#returning-values)
* [Summing Up](https://cs50.harvard.edu/python/2022/notes/0/#summing-up)

[**Creating Code with Python**](https://cs50.harvard.edu/python/2022/notes/0/#creating-code-with-python)

* VS Code is a special type of text editor that is called a compiler. At the top, you’ll notice a text editor and, at the bottom you will see a terminal where you can execute commands.
* In the terminal, you can execute code hello.py to start coding.
* In the text editor above, you can type print("hello, world"). This is a famous canonical program that nearly all coders write during their learning process.
* In the terminal window, you can execute commands. To run this program, you are going to need to move your cursor to the bottom of the screen, clicking in the terminal window. You can now type a second command in the terminal window. Next to the dollar sign, type python hello.py and press the enter key on your keyboard.
* Recall, computers really only understand zeros and ones. Therefore, when you run python hello.py, python will interpret the text that you created in hello.py and translate it into the zeros and ones that the computer can understand.
* The result of running the python hello.py program is hello, world.
* Congrats! You just created your first program.

[**Functions**](https://cs50.harvard.edu/python/2022/notes/0/#functions)

* Functions are verbs or actions that the computer or computer language will already know how to perform.
* In your hello.py program, the print function knows how to print to the terminal window.
* The print function takes arguments. In this case, "hello, world" are the arguments that the print function takes.

[**Bugs**](https://cs50.harvard.edu/python/2022/notes/0/#bugs)

* Bugs are a natural part of coding. These are mistakes, problems for you to solve! Don’t get discouraged! This is part of the process of becoming a great programmer.
* Imagine in our hello.py program that accidentally typed print("hello, world" notice that we missed the final ) required by the compiler. If I purposefully make this mistake, you’ll the compiler will output an error in the terminal window!
* Often, the error messages will inform you of your mistake and provide you clues on how to fix them. However, there will be many times that the compiler is not this kind.

[**Improving Your First Python Program**](https://cs50.harvard.edu/python/2022/notes/0/#improving-your-first-python-program)

* We can personalize your first Python program.
* In our text editor in hello.py we can add another function. input is a function that takes a prompt as an argument. We can edit our code to say
* input("What's your name? ")
* print("hello, world")
* This edit alone, however, will not allow your program to output what your user inputs. For that, we will need to introduce you to variables

[**Variables**](https://cs50.harvard.edu/python/2022/notes/0/#variables)

* A variable is just a container for a value within your own program.
* In your program, you can introduce your own variable in your program by editing it to read
* name = input("What's your name? ")
* print("hello, world")

Notice that this equal = sign in the middle of name = input("What's your name? ") has a special role in programming. This equal sign literally assigns what is on the right to what is on the left. Therefore, the value returned by input("What's your name? ") is assigned to name.

* If you edit your code as follows, you will notice an error
* name = input("What's your name? ")
* print("hello, name")
* The program will return hello, name in the terminal window regardless of what the user types.
* Further editing our code, you could type
* name = input("What's your name? ")
* print("hello,")
* print(name)
* The result in the terminal window would be
* What's your name? David
* hello
* David
* We are getting closer to the result we might intend!
* You can learn more in Python’s documentation on [data types](https://docs.python.org/3/library/datatypes.html).

[**Comments**](https://cs50.harvard.edu/python/2022/notes/0/#comments)

* Comments are a way for programmers to track what they are doing in their programs and even inform others about their intentions for a block of code. In short, they are notes for yourself and others that will see your code!
* You can add comments to your program to be able to see what it is that your program is doing. You might edit your code as follows:
* *# Ask the user for their name*
* name = input("What's your name? ")
* print("hello,")
* print(name)
* Comments can also serve as to-do list for you.

[**Pseudocode**](https://cs50.harvard.edu/python/2022/notes/0/#pseudocode)

* Pseudocode is an important type of comment that becomes a special type of to-do list, especially when you don’t understand how to accomplish a coding task. For example, in your code, you might edit your code to say:
* *# Ask the user for their name*
* name = input("What's your name? ")
* *# Print hello*
* print("hello,")
* *# Print the name inputted*
* print(name)

[**Further Improving Your First Python Program**](https://cs50.harvard.edu/python/2022/notes/0/#further-improving-your-first-python-program)

* We can further edit our code as follows:
* *# Ask the user for their name*
* name = input("What's your name? ")
* *# Print hello and the inputted name*
* print("hello, " + name)
* It turns out that some functions take many arguments.
* We can use a comma , to pass in multiple arguments by editing our code as follows:
* *# Ask the user for their name*
* name = input("What's your name? ")
* *# Print hello and the inputted name*
* print("hello,", name)

The output in the terminal, if we typed “David” we would be hello, David. Success.

[**Strings and Paremeters**](https://cs50.harvard.edu/python/2022/notes/0/#strings-and-paremeters)

* A string, known as a str in Python, is a sequence of text.
* Rewinding a bit in our code back to the following, there was a visual side effect of having the result appear on multiple lines:
* *# Ask the user for their name*
* name = input("What's your name? ")
* print("hello,")
* print(name)
* Functions take arguments that influence their behavior. If we look at the documentation for [print](https://docs.python.org/3/library/functions.html#print) you’ll notice we can learn a lot about the arguments that the print function takes.
* Looking at this documentation, you’ll learn that the print function automatically include a piece of code end='\n'. This \n indicates that the print function will automatically create a line break when run. The print function takes an argument called end` and the default is to create a new line.
* However, we can technically provide an argument for end ourselves such that a new line is not created!
* We can modify our code as follows:
* *# Ask the user for their name*
* name = input("What's your name? ")
* print("hello,", end="")
* print(name)

By providing end="" we are over-writing the default value of end such that it never creates a new line after this first print statement. Providing the name as “David”, the output in the terminal window will be hello, David.

* Parameters, therefore, are arguments that can be taken by a function.
* You can learn more in Python’s documentation on [print](https://docs.python.org/3/library/functions.html#print).

[**A small problem with quotation marks**](https://cs50.harvard.edu/python/2022/notes/0/#a-small-problem-with-quotation-marks)

* Notice how adding quotation marks as part of your string is challenging.
* print("hello,"friend"") will not work and the compiler will throw an error.
* Generally, there are two approaches to fixing this. First, you could simply change the quotes to single quote marks.
* Another, more commonly used approach would be code as print("hello, \"friend\""). The backslashes tell the compiler that the following character should be considered a quotation mark in the string and avoid a compiler error.

[**Formatting Strings**](https://cs50.harvard.edu/python/2022/notes/0/#formatting-strings)

* Probably the most elegant way to use strings would be as follows:
* *# Ask the user for their name*
* name = input("What's your name? ")
* print(f"hello, **{**name**}**")

Notice the f in print(f"hello, {name}"). This f is a special indicator to Python to treat this string a special way, different than previous approaches we have illustrated in this lecture. Expect that you will be using this style of strings quite frequently in this course.

[**More on Strings**](https://cs50.harvard.edu/python/2022/notes/0/#more-on-strings)

* You should never expect your user will cooperate as intended. Therefore, you will need to ensure that the input of your user is corrected or checked.
* It turns out that built into strings is the ability to remove whitespace from a string.
* By utilizing the method strip on name as name = name.strip(), it will strip all the whitespaces on the left and right of the users input. You can modify your code to be:
* *# Ask the user for their name*
* name = input("What's your name? ")
* *# Remove whitespace from the str*
* name = name.strip()
* *# Print the output*
* print(f"hello, **{**name**}**")

Rerunning this program, regardless of how many spaces you type before or after the name, it will strip off all the whitespace.

* Using the title method, it would title case the user’s name:
* *# Ask the user for their name*
* name = input("What's your name? ")
* *# Remove whitespace from the str*
* name = name.strip()
* *# Capitalize the first letter of each word*
* name = name.title()
* *# Print the output*
* print(f"hello, **{**name**}**")
* By this point, you might be very tired of typing python repeatedly in the terminal window. You cause us the up arrow of your keyboard to recall the most recent terminal commands you have made.
* Notice that you can modify your code to be more efficient:
* *# Ask the user for their name*
* name = input("What's your name? ")
* *# Remove whitespace from the str and capitalize the first letter of each word*
* name = name.strip().title()
* *# Print the output*
* print(f"hello, **{**name**}**")

This creates the same result as your previous code.

* We could even go further!
* *# Ask the user for their name, remove whitespace from the str and capitalize the first letter of each word*
* name = input("What's your name? ").strip().title()
* *# Print the output*
* print(f"hello, **{**name**}**")
* You can learn more about strings in Python’s documentation on [str](https://docs.python.org/3/library/stdtypes.html#str)

[**Integers or int**](https://cs50.harvard.edu/python/2022/notes/0/#integers-or-int)

* In Python, an integer is referred to as an int.
* In the world of mathematics, we are familiar with +, -, \*, /, and % operators. That last operator % or modulo operator may not be very familiar to you.
* You don’t have to use the text editor window in your compiler to run Python code. Down in your terminal, you can run python alone. You will be presented with >>> in the terminal window. You can then run live, interactive code. You could type 1+1 and it will run that calculation. This mode will not commonly be used during this course.
* Opening up VS Code again, we can type code calculator.py in the terminal. This will create a new file in which we will create our own calculator.
* First, we can declare a few variables.
* x = 1
* y = 2
* z = x + y
* print(z)

Naturally, when we run python calculator.py we get the result in the terminal window of 3. We can make this more interactive using the input function.

x = input("What's x? ")

y = input("What's y? ")

z = x + y

print(z)

* Running this program, we discover that the output is incorrect as 12. Why might this be?
* Prior, we have seen how the + sign concatenates two strings. Because your input from your keyboard on your computer comes into the compiler as text, it is treated a string. We, therefore, need to convert this input from a string to an integer. We can do so as follows:
* x = input("What's x? ")
* y = input("What's y? ")
* z = int(x) + int(y)
* print(z)

The result is now correct. The use of int(x), is called “casting” where a value is temporarily changed from one type of variable (in this case a string) to another (here, an integer).

* We can further improve our program as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* print(x + y)

This illustrates that you can run functions on functions. The most inner function is run first, and then the outer one is run. First, the input function is run. Then, the int function.

* You can learn more in Python’s Documenation of [int](https://docs.python.org/3/library/functions.html?highlight=float#int).

[**Readability Wins**](https://cs50.harvard.edu/python/2022/notes/0/#readability-wins)

* When deciding on your approach to a coding task, remember that one could make a reasonable argument for many approaches to the same problem.
* Regardless of what approach you take to a programming task, remember that your code must be readable. You should use comments to give yourself and others clues about what your code is doing. Further, you should create code in a way that is readable.

[**Float Basics**](https://cs50.harvard.edu/python/2022/notes/0/#float-basics)

* A floating point value is a real number that has a decimal point in it, such as 0.52.
* You can change your code to support floats as follows:
* x = float(input("What's x? "))
* y = float(input("What's y? "))
* print(x + y)

This change allows your user to enter 1.2 and 3.4 to present a total of 4.6.

* Let’s imagine, however, that you want to round the total to the nearest integer. Looking at the Python documentation for round you’ll see that the available arguments are round(number[n, ndigits]). Those square brackets indicate that something optional can be specified by the programmer. Therefore, you could do round(n) to round a digit to its nearest integer. Alternatively, you could code as follows:
* *# Get the user's input*
* x = float(input("What's x? "))
* y = float(input("What's y? "))
* *# Create a rounded result*
* z = round(x + y)
* *# Print the result*
* print(z)

The output will be rounded to the nearest integer.

* What if we wanted to format the output of long numbers? For example, rather than seeing 1000, you may wish to see 1,000. You could modify your code as follows:
* *# Get the user's input*
* x = float(input("What's x? "))
* y = float(input("What's y? "))
* *# Create a rounded result*
* z = round(x + y)
* *# Print the formatted result*
* print(f"**{**z**:**,**}**")

Though quite cryptic, that print(f"{z:,}") creates a scenario where the outputted z will include commas where the result could look like 1,000 or 2,500.

[**More on Floats**](https://cs50.harvard.edu/python/2022/notes/0/#more-on-floats)

* How can we round floating point values? First, modify your code as follows:
* *# Get the user's input*
* x = float(input("What's x? "))
* y = float(input("What's y? "))
* *# Calculate the result*
* z = x / y
* *# Print the result*
* print(z)

When inputting 2 as x and 3 as y, the result z is 0.6666666666 seemingly going on to infinite as we might expect.

* Let’s imagine that we want to round this down, we could modify our code as follows:
* *# Get the user's input*
* x = float(input("What's x? "))
* y = float(input("What's y? "))
* *# Calculate the result and round*
* z = round(x / y, 2)
* *# Print the result*
* print(z)

As we might expect, this will round the result to the nearest two decimal points.

* We could also use fstring to format the output as follows:
* *# Get the user's input*
* x = float(input("What's x? "))
* y = float(input("What's y? "))
* *# Calculate the result*
* z = x / y
* *# Print the result*
* print(f"**{**z**:**.2f**}**")

This cryptic fstring code displays the same as our prior rounding strategy.

* You can learn more in Python’s documentation of [float](https://docs.python.org/3/library/functions.html?highlight=float#float).

[**Def**](https://cs50.harvard.edu/python/2022/notes/0/#def)

* Wouldn’t it be nice to create our own functions?
* Let’s bring back our final code of hello.py by typing code hello.py into the terminal window. Your starting code should look as follows:
* *# Ask the user for their name, remove whitespace from the str and capitalize the first letter of each word*
* name = input("What's your name? ").strip().title()
* *# Print the output*
* print(f"hello, **{**name**}**")

We can better our code to create our own special function that says “hello” for us!

* Erasing all our code in our text editor, let’s start from scratch:
* name = input("What's your name? ")
* hello()
* print(name)

Attempting to run this code, your compiler will throw an error. After all, there is no defined function for hello.

* We can create our own function called hello as follows:
* **def** hello():
* print("hello")
* name = input("What's your name? ")
* hello()
* print(name)

Notice that everything under def hello() is indented. Python is an indented language. It uses indentation to understand what is part of the above function. Therefore, everything in the hello function must be indented. When something is not indented, it treats it as if it is not inside the hello function. Running python hello.py in the terminal window, you’ll see that your output is not exactly as you may want.

* We can further improve our code:
* *# Create our own function*
* **def** hello(to):
* print("hello,", to)
* *# Output using our own function*
* name = input("What's your name? ")
* hello(name)

Here, in the first lines, you are creating your hello function. This time, however, you are telling the compiler that this function takes a single parameter: a variable called to. Therefore, when you call hello(name) the computer passes name into the hello function as to. This is how we pass values into functions. Very useful! Running python hello.py in the terminal window, you’ll see that the output is much closer to our ideal presented earlier in this lecture.

* We can change our code to add a default value to hello:
* *# Create our own function*
* **def** hello(to="world"):
* print("hello,", to)
* *# Output using our own function*
* name = input("What's your name? ")
* hello(name)
* *# Output without passing the expected arguments*
* hello()

Test out your code yourself. Notice how the first hello will behave as you might expect and the second hello, which is not passed a value, will by default output hello, world.

* We don’t have to have our function at the start of our program. We can move it down, but we need to tell the compiler that we have a main function and we have a separate hello function.
* **def** main():
* *# Output using our own function*
* name = input("What's your name? ")
* hello(name)
* *# Output without passing the expected arguments*
* hello()
* *# Create our own function*
* **def** hello(to="world"):
* print("hello,", to)

This alone, however, will create an error of sorts. If we run python hello.py nothing happens! The reason for this is that nothing in this code is actually calling the main function and bringing our program to life.

* The following very small modification will call the main function and restore our program to working order:
* **def** main():
* *# Output using our own function*
* name = input("What's your name? ")
* hello(name)
* *# Output without passing the expected arguments*
* hello()
* *# Create our own function*
* **def** hello(to="world"):
* print("hello,", to)
* main()

[**Returning Values**](https://cs50.harvard.edu/python/2022/notes/0/#returning-values)

* You can imagine many scenarios where you don’t just want a function to perform an action, but also to return a value back to the main function. For example, rather than simply printing the calculation of x + y, you may want a function to return the value of this calculation back to another part of your program. This “passing back” of a value we call a return value.
* Returning to our calculator.py code by typing code calculator.py. Erase all code there. Rework the code as follows:
* **def** main():
* x = int(input("What's x? "))
* print("x squared is", square(x))
* **def** square(n):
* **return** n \* n
* main()

Effectively, x is passed to square. Then, the calculation of x \* x is returned back to the main function.

[**Summing Up**](https://cs50.harvard.edu/python/2022/notes/0/#summing-up)

Through the work of this single lecture, you have learned abilities that you will use countless times in your own programs. You have learned about…

* Creating your first programs in Python;
* Functions;
* Bugs;
* Variables;
* Comments;
* Pseudocode;
* Strings;
* Parameters;
* Formatted Strings;
* Integers;
* Principles of readability;
* Floats;
* Creating your own functions; and
* Return values.

[GENTLE ORCHESTRAL MUSIC PLAYING]

DAVID MALAN: All right, this is CS50's Introduction

to Programming with Python.

My name is David Malan and this is our week on functions and variables.

But odds are, many of you, most of you, have never actually programmed before.

So let's start by doing just that.

Let me go ahead here and open up my computer

and on it, a program called Visual Studio Code or VS

Code, which is just a very popular program nowadays

for actually writing code.

Now, you don't have to write code using this particular tool.

In fact, all we need at the end of the day

is a so-called text editor, a program for writing text.

And, heck, if you really want, you could even use something

like Google Docs or Microsoft Word.

You'd have to save it in the right format,

but really, at the end of the day, all you need is a program for writing text

because that's what code is-- text.

Now within this particular program, I'm going

to have the ability to create one or more files

via this top portion of the screen.

And I'm going to do so by diving right in

and doing this at the bottom of my screen.

At the bottom of my screen is a so-called terminal window

and this is a command line interface or CLI interface to the underlying

computer, be it your Mac or your PC or even some server in the cloud.

And what I'm going to do here is literally write, "code" and then

the name of the file that I want to code, for instance, Hello.py.

As we'll soon see, any program that you write in Python generally has a file

name that ends in .py to indicate to the computer that it's indeed a program

written in Python.

Now you'll see here at the top of my screen, I have a blinking cursor,

a line one-- which is where the very first line of my code is going to go--

and then just a tab that reminds me of the name of this file Hello.py.

And without even knowing much Python, I'm

going to write my very first program here as follows--

print("hello,world").

And you'll see that, at my keyboard, some of my thoughts

were finished for me.

I only had to type one parenthesis and the other one automatically appeared,

and that's just the feature that we'll see of tools like this tool here.

Now even if you've never programmed before,

odds are you can guess, infer what this simple program is going to do.

And it's only one line. print("hello,world").

Indeed, when I run this program ultimately,

it's just going to say, Hello to the world.

And, in fact, this is a very famous--

perhaps the most canonical-- program that you

can write as your very first program in Python or any other language,

and so that's what I've done here.

But on my Mac, my PC, even my phone, I'm generally in the habit,

like you, of running programs by double clicking an icon

or just tapping on the screen.

But I see no such icons here.

And in fact, that's because my interface to at least my current Mac

or PC or some server in the cloud, is again only a CLI--

command line interface-- which even though it

might feel like it's a step back from the menus and buttons and icons

that you and I take for granted every day, you'll find, we think,

that it's ultimately a much more powerful interface and incredibly

popular to use among programmers in the real world.

So to run this program, I'm going to have to use a command

and I'm going to move my cursor back down to the bottom of the screen

here where previously I already ran one command--

the command Code which has the effect of opening VS Code in my computer,

and then I passed in the name of the file that I wanted to code up.

Now I have a chance to type a second command,

and you'll see I see a second dollar sign.

Now the dollar sign here doesn't indicate any kind of currency or money,

it just is the symbol that's generally used

to indicate your prompt-- where the command line interface wants

you to put those commands.

Now the command I can run here is going to be this--

I'm going to run Python of Hello.py.

Now why is that?

Well, it turns out that when I actually write code in a language like Python,

it's of course stored in that file--

Hello.py, but I need to interpret the code top to bottom,

left to right so that the computer knows what to do.

Indeed, at the end of the day, even if you don't really

know much about computers, you've probably

heard the computers only understand zeros and ones, the so-called binary

system.

Well if that's the case, then something that says "print" and parentheses

and quote unquote hello world, is not surely zeros and ones.

We have to somehow translate it into the zeros and ones

that the computer understands.

Now fortunately, so long as you've installed such a program in advance,

there's a program as well as a language called Python.

So Python is not only a language in which we're going to write code,

it's also a program otherwise known as an interpreter

that you install for free on your own Mac or PC or some server in the cloud

and you can then run that program, that interpreter, passing to it as input

the name of your file like mine here, Hello.py.

And then that program, that interpreter, will

handle the process of reading it top to bottom, left to right

and translating it effectively into those zeros and ones

that the computer can understand.

So let's do just that.

Let me go back to VS Code here.

I already typed out Python of Hello.py but I didn't get hit Enter.

And that's what's now going to kick off this command

and hopefully, if I didn't mess any of this up,

I should see my very first program's output to the screen.

And voila, hello, world.

So if you too have typed exactly that same code

and have executed exactly that same command,

you will have written your very first program in this case in Python.

Well now let's take a step back and consider,

what is it that we actually just did and what

is it we're looking here on the screen?

Well first and foremost, in most any programming language,

you tend to have access to what are called functions.

A function is like an action or a verb that

lets you do something in the program.

And generally speaking, any language comes with some predetermined set

of functions-- some very basic actions or verbs that the computer will already

know how to do for you, that the language, really,

will know how to do for you.

And you, the programmer, the human, can use those functions at will

to get the computer to do those things.

Now the program in question here, Hello.py is using one function.

And you could perhaps guess what it is.

That function is, of course, going to be this function print

and that print function, of course, doesn't

print some preordained string of text.

That is to say, it prints whatever it is you want it to print.

And here too, do we have another piece of terminology in the world

of programming, namely arguments.

An argument is an input to a function that somehow influences its behavior.

The people who invented Python, of course,

didn't necessarily know what it is you and I

are going to want to print to the screen, so they designed this print

function using these parentheses with the ability

to take as input some string of text, be it

in English or any other human language, that is

what you want this function ultimately to print onto the screen.

And what is it that the program is ultimately doing on the screen?

Well, it's printing, of course.

It's showing us hello, world on the screen.

And that's generally in programming known as a side effect.

It can be visual, it can be audio.

In this case, it's something that appears on the screen

and functions, therefore, can indeed have these side effects.

One of the things they can do as this verb or action is

to display on the screen as a side effect, something like those words

that we wanted--

hello, world.

So that's my first program and, you know, I'm feeling pretty good.

Everything worked as planned.

I didn't make any mistakes, but, honestly,

when you're learning how to program and even

once you've learned how to program years later, you're going to make mistakes.

And those mistakes, of course, are referred

to a term you might already know, which is that of a bug.

A bug is a mistake in a program and they can take so many forms.

And take comfort, perhaps, in knowing that over the coming weeks,

you're going to make so many mistakes.

You're going to have so many bugs in your code, just like I did

and just as I still do.

And those bugs themselves are just mistakes

that are problems for you to solve.

And over the weeks to come, we're going to give you

a lot of tools both mental and technical via which you can solve those problems.

But just don't get discouraged if when writing your program for the first time

it doesn't even work that first time, it will with time, with practice,

and with experience.

So let me deliberately now make a mistake

that there was a non-zero chance I might have done accidentally already,

but I got lucky.

Let me go ahead and just suppose I forgot

to include something like the closing parenthesis

at the end of this line of code.

The code is almost correct, it's like 99% of the way there,

but now that I've pointed it out, it's pretty obvious

that it's missing that closed parenthesis.

But even little, seemingly minor details like that, that you and I as humans

wouldn't really care about.

And if you're sending an email or a text message, whatever, it's just a typo.

It's not that big a deal.

It is going to be a big deal to the computer.

A computer is going to take you literally,

and if you don't finish your thought in the way the language expects,

it's not going to necessarily run at all.

So let's do this.

I'm going to go ahead here and clear my screen down at the bottom

just so I can start fresh, and I'm going to go ahead and run

this version of my program after having made that change by deleting

the parenthesis.

I'm going to go ahead and type Python again of Hello.py and this time

when I hit Enter, I'm hoping I'm going to see hello, world,

but here we have an error on the screen, a so-called syntax

error, which refers to my having made a mistake at my keyboard.

And this one, fortunately, is pretty straightforward.

It indeed says that this open parenthesis was never closed.

And so that's probably pretty intuitive.

Now what I need to do, I need to, of course, to close it.

Unfortunately, sometimes the error messages we'll see in the coming weeks

are not going to be nearly that user-friendly.

But there too again, with experience, with practice, will you

get better at debugging such programs.

Let me now make sure that I indeed fixed it correctly.

Let me go ahead and run now Hello.py and hit Enter and, voila,

we're back in business.

Well let me pause here and see if we have any questions now

about Python itself, writing, or running even the simplest of these programs.

AUDIENCE: Could I write code inside a word or, for example, Microsoft Excel?

And what's the barrier to doing that?

DAVID MALAN: A really good question, and allow

me to very explicitly say to the entire internet

that you should not write code with Microsoft Word.

I mentioned that only because it's a tool via which

you can write text and code is, at the end of the day, just text.

But it's not the right tool for the job.

We don't need bold facing, underlining, paragraphs and the like.

We generally want something much simpler than Microsoft Word or Google Docs.

And so VS Code is an example of just a more general purpose text editor.

Its purpose in life is to allow you, the human, to edit text.

Nowadays these text editors come with many more features.

In fact, you'll notice that even in my code here,

even though it's just one line, there's a bit of color to it.

The word "print" for me is appearing in blue.

The parentheses are black.

And we'll see as we write more lines of code, more and more of the lines

will come to life in various colors.

Now that's just one feature of a text editor.

We'll see too that it has features like this built-in terminal window.

It's going to have a built-in tool for debugging or finding problems

with code.

And it's just a very popular tool nowadays,

but there are many, many others out there.

You're welcome to use them for this course and beyond.

We just happen to use this one in large part

too because you can also use VS Code nowadays for free in the cloud.

How about one other question here on programming with Python

or hello, world or syntax more generally?

AUDIENCE: Yeah I was trying to ask if it's not

possible to run the computer using the terminal window?

DAVID MALAN: I think I heard is it not--

if it's possible to run the program without the terminal window?

Are you--

AUDIENCE: Yes, sir.

DAVID MALAN: OK, you froze for me again.

But let me infer what the question is.

So in this environment, as I've configured my computer,

I can only run these Python programs via the terminal window.

Now that's good for me, the programmer, or the person who's

trying to learn how to program, but it's not very good

if you want to ship this software and have other people use your actual code.

You can absolutely write programs and then allow other people to use,

not a command line interface, but a graphical user interface or GUI--

G-U-I. This is just one mechanism and perhaps, I think,

the best one with which to start writing code

because eventually it's going to give us a lot more control.

Allow me to forge ahead here, but please feel

free to continue asking questions along the way if only via the chat.

Let's consider now how we might go about improving this program.

Let's go about improving this program to make it a little more interactive

and not just assume that everyone is going to want to be greeted

more generically as hello, world.

Let's see if I can't get this program to say something like Hello, David

or Hello, Jeremiah or Hello, Horatio or whatever the actual user's name is.

Well to do this I'm going to go back up to Hello to pi

and I'm going to add another line of code at the very top that simply says,

for instance, what's your name, quote unquote with an extra space at the end.

So I'm printing to the user asking them a question for some input,

but now I need another function to actually get input from the user.

And, perfectly, enough Python comes with a function named input.

So here I'm going to go ahead and call a function input, open paren,

close paren.

And that's going to prompt the user with just a blinking cursor waiting

for them to type something in.

Now it turns out, if I read the documentation for the input function,

it actually takes an argument itself.

I don't need to use "print" separately and then prompt the user for input.

So I can actually simplify this code before we even use it.

I'm going to go ahead here and take that same string from print,

put it as an argument to the input function,

and get rid of the "print" altogether.

And, in fact, that "print" would have added a new line anyway.

So now I've just got a prompt where the user's cursor is

going to end up blinking at the end of the line asking them, what's your name?

In my terminal window I'm going to run Python of Hello.py, Enter.

OK.

We're making progress.

It seems that this new function input is indeed prompting me, the human,

for input.

So I'm going to type in my name--

David-- and hit Enter.

Unfortunately, it doesn't really do anything with my name,

it just outputs it immediately.

All right, well, I could fix this, right?

I could go up to line 2 and I could change "world" to "David,"

and then back in my terminal window here I can do Python of Hello.py, Enter.

What's your name?

David, Enter.

And there we go.

All right now I'm up and running.

Now my program is working as intended.

Of course, this isn't really working as intended here.

Let me go ahead and try pretending to be my colleague, Carter here.

Well Carter's name is this.

I'm going to go ahead and hit Enter and I'll see, of course, Hello, Carter--

well, obviously not, because I've hardcoded, so to speak,

I've written literally my name inside of the string.

So we need some way now of actually getting back what the user's input is

and doing something with it ultimately.

And for this we're going to leverage another feature of programming,

specifically a feature of some functions, which is that they

can have return values as well.

If you think of input as being, again, this action,

this verb-- you can actually personify it as maybe a person,

like a friend of yours that you've asked a question of

and you've asked your friend to go get input from someone else--

go ask that person their name.

And if your friend comes back knowing that person's name, well,

wouldn't it be nice if they handed that name back to you?

That's kind of what we need metaphorically the function to do,

is get the user's input and then hand it back to me so that I, the programmer,

can do something with it.

But if it's going to be handed back to me,

I kind of want to put it somewhere so that I can then print it back

on the screen.

I need to do the equivalent of take out like a piece of paper or Post-It note,

write down on this piece of paper what it is the human has said,

so that I can then feed it into as input that print function.

And to do that, we're going to need one more feature of programming, namely

variables.

And odds are, most everyone's familiar with variables from math class

way back when-- x and y and z and the like.

Well, programming has that same capability, this ability

to create a variable-- in this case, in the computer's memory--

not just on a piece of paper.

And that variable can store a value--

a number, some text, even an image or video or more.

A variable is just a container for some value inside of a computer

or inside of your own program.

So how do I go about expressing myself in this way?

Well I think what I'm going to do is introduce

a variable that's a little more interestingly named than x or y.

I could just say this-- x = input.

But I'm going to use a better name than a typical mathematical variable here

and I'm going to literally call my variable "name," why?

Well in programming, because I have a whole keyboard in front of me,

I can use more descriptive terms to describe what it is I'm writing.

And now, though, there's an opportunity to consider a specific piece of syntax.

We've seen parentheses, we've seen quotes, all of which

are necessary when passing inputs to a function,

but this equal sign here that's in between input on the right and name

on the left is actually important.

And it's technically not an equal sign per se.

It doesn't mean equality as much as it means assignment.

So in Python and many programming languages,

a single equal sign is the assignment operator

and what that means specifically is that, you

want to assign from right to left whatever the user's input is.

So the equal sign copies from the right to the left whatever the return

value of the function on the right is.

So again, the input function clearly gets input from the user.

That's why I was able to type my name or Carter's, but it also

sort of behind the scenes hands that value, that return value, back to me

the programmer.

And if I use an equal sign and a variable, no matter what I call it,

I can store that input in that variable so as to reuse it later.

So now, sitting in the computer's memory somewhere,

is a container containing "David" quote, unquote or "Carter" quote,

unquote, or whatever the human has typed in.

But here it's easy to make a mistake.

Suppose I decide to try to print that name and so

I kind of on a hunch type in this, hello, name.

Just kind of plugging in the name of the variable.

Well let me go ahead here and run Python of Hello.py and hit Enter.

That's going to prompt me for my name.

And let me type in my name--

D-A-V-I-D. But I haven't hit Enter yet.

And perhaps via the chat, what's going to happen here when I now hit Enter?

I'm hoping it says, Hello, David.

I'd be OK if it says Hello, world, but I don't want it to say what it's actually

going to say and, yep, what we're seeing in the chat is,

well, it's probably going to say literally Hello, name.

So that's not quite right.

So we need another way of printing out the value inside of that variable

rather than just this word "name."

Well let me try this in a couple of different ways.

Let me try this as follows--

let me go ahead and maybe undo this because I've gotten pretty good already

at saying Hello.

So let's draw that line in the sand and just say, all right,

let's get at least get Hello comma out the door.

Let's now print name and just on a hunch, I'm going to try this.

I'm going to use print again--

because you can use these functions as many times as you need--

and I'm going to pass to the print function the variable called name.

But notice I'm being a little clever now.

I'm not putting it in double quotes because we've seen already that double

quotes means literally print out N-A-M-E.

I'm getting rid of the quotes this time in hopes that,

now by passing the variable called name to the function called print it will,

in fact, go about printing the contents of that variable that is,

its so-called value.

All right, let's go ahead and do this here.

Python of Hello.py, Enter.

What's your name?

David.

And now, crossing my fingers still, I see Hello comma David.

All right.

So it's not the best program.

I'm kind of cutting some corners here, so to speak.

I'm saying Hello, David on two separate lines.

So it's not as elegant, it's not as pretty,

it's not as grammatically appropriate in English as just saying it

all in one breath on one line, but at least I've solved the problem, just not

very well yet.

But let me take a step back now and perhaps introduce

a couple of other concepts with which we should be familiar,

which is as our programs get longer and they're no longer just one line or two

or even three, eventually our programs are going to become dozens of lines,

maybe even hundreds of lines long.

Let's set the stage for success moving forward.

It turns out that Python and a lot of programming languages

also support something called comments.

Comments are notes to yourself in your code

and you include comments by way of a special symbol-- in Python

it's going to be the hash symbol, typically--

and that allows you to write the equivalent

of a note to yourself but in a way that's not going to break your code.

The computer actually ignores your comment.

It's just there for you, it's just there for your teacher,

it's just there for your colleague with whom

you're sharing ultimately that code.

So if I go back to VS Code here and I just

want to add some comments to this program to explain to my teacher,

to myself, to my colleagues what this program is doing,

well, let's go ahead and do that.

I'm going to go at the very top of my program and on line 1

now I'm going to move that original line of code down a bit,

I'm going to add a hash, and I'm going to say something like this,

ask user for their name.

Now, I don't have to use that language.

I don't have to use that text.

I could use any human language whatsoever.

It doesn't have to be English.

But I'm going to now, below that, just say something like this--

say Hello to user.

And you'll notice that VS Code by default

is kind of graying out my comments.

They're no longer blue, there's no red, there's no color in them.

And that's just because they're notes to myself

and the computer ultimately is going to ignore them.

But what we have now is two comments-- ask user

for their name and then a second comment, say Hello to user.

And I've just kind of commented each chunk of code,

like each line or lines plural of code, that are doing something noteworthy.

Why?

Well, tomorrow morning when I wake up having

slept for quite some time, forgotten what it is I did the previous day,

it's convenient with comments to just see

in English or your own human language what

it is this program is doing so that you don't have to read the code itself

and, better yet, if there's maybe a mistake down the road,

you can read what your intention was and then you can look at the code

and figure out if your code is now doing what you intended.

So this isn't really necessary for a program this small.

It's pretty obvious with just one or two or three lines

what the program is doing.

It's just as fast to read the code than the comments,

but getting into this habit is generally a good thing--

to comment your code every one or a few lines so as to remind yourself

and others what it is your intent and your code is doing.

What's nice about comments too is this--

comments can also serve to be sort of a to-do list for yourself.

There's this notion in programming of pseudocode.

Pseudocode isn't a formal thing.

It's not one specific language.

It's just using English or your own human language

to express your thoughts succinctly, methodically, algorithmically,

so to speak.

But pseudocode, therefore, because it's not Python

and it's not necessarily English, it just kind of allows

you to outline your program even in advance.

So for instance, if I wasn't sure today how

I wanted to go about writing this program

but I didn't know what I wanted to do, I could have started today

by just writing this in Hello.py.

No code, I could have written just a couple of comments to myself--

Step 1, ask user for their name.

Step 2, say Hello to user.

Then once I've outlined my program in pseudocode, then I can go in there

and say, all right, how do I ask the user for their name?

Well, I can do input "what's your name" question?

And then on the left here, I can maybe put a variable and assign it to that.

OK, how do I say Hello to the user?

Well, I know I can use print to say things on the screen.

Let me say hello, and let me--

OK, let me now print the person's name.

So again, pseudocode is a nice way of structuring your to-do list,

especially if you have no idea how to write the code,

because it breaks a bigger program down into small bite-sized tasks.

All right, let me pause here to see if there are now

any questions on comments, pseudocode, return values, or variables.

Any questions we can clear up here?

AUDIENCE: Yeah, my question is does the function input

work for any type of information, or only for words?

DAVID MALAN: Yeah, really good question.

So according to its documentation, and we'll look more at formal documentation

soon, input is going to expect what's called

a string, that is a sequence of text, be it

in English or any other human language.

But it's indeed going to be expecting text with which to prompt the user.

A good question.

How about another question from the group, if we could?

AUDIENCE: I wanted to ask how would I make a several line comment?

DAVID MALAN: Oh, how do you do many lines of comments,

if I'm hearing you correctly?

AUDIENCE: Yes.

DAVID MALAN: Sure.

You would just keep doing them like this.

You just prefix each of the lines with a hash symbol, like I'm doing here.

There is another technique for doing multi-line comments in Python that

actually tend to have special meaning.

You can do three double quotes like this and then anything in between here

is a comment.

That's another technique.

Or you can use single quotes as well.

But more on those, I think, another time.

All right, well, you don't mind, let me forge ahead here

and see how we might improve this program further

and also introduce a few other features that we might

want to take into account over time.

So it turns out that we can certainly improve on this program

because it's a little disappointing that I'm cutting this corner

and saying Hello comma and then on a new line printing out name.

Like we can do better.

And most programs you use on your phone or your laptop certainly

keep text together when people want.

So how can we go about doing that?

Well there's a few different ways.

And in fact, the goal here is not so much to solve this one problem,

but to demonstrate and emphasize that in programming--

Python and other languages-- there's so many ways

sometimes to solve the same problem.

And here's one way to solve this problem.

Let me go in here and let me go ahead now and say hello,

and let me just add to the end of that the user's name.

So I'm using + in kind of an interesting way.

This is not addition, per se.

I'm not adding numbers, obviously, but I do

kind of want to add the person's name to the string of text Hello comma.

Well let me go now down to my terminal window and run Python of Hello.py

again, Enter.

What's your name?

I'm going to type in David.

Enter.

OK, it's better.

It's better, but there's a minor bug, albeit aesthetic here.

There's missing space, but let's just use some intuition here.

Well, if I'm missing the space after the comma,

why don't I go ahead and just add it manually here.

Let me now rerun the program Python of Hello.py, Enter, David, Enter.

And there we go.

Now we have something that looks a little prettier

in terms of English grammar.

Hello, comma, space, David.

And now if we rewind, you might have noticed before

or wondered why I had this seemingly extra space after my question mark,

namely here.

There's a space after the question mark but before the double quote

and that was just for aesthetics too.

I wanted to move the user's cursor one space to the right

so that when I typed their name or they typed their name,

it's not immediately next to that same question mark there.

But there's other ways we can do this.

It turns out that some functions, print among them,

actually take multiple arguments.

And it turns out that if you separate the inputs

to a function-- the so-called arguments to a function-- with a comma,

you can pass in not just one, but two, three, four, five, onward.

So let me go ahead and pass in not just hello, comma, space, but that followed

by name.

And this is a little confusing potentially at first glance

because now I've got two commas but it's important to note

that the first comma is inside of my quotation marks, which is simply

an English grammatical thing.

The second comma here is outside of the quotes,

but between what are now two separate arguments to print.

The first argument is hello, comma, space.

The second argument is the name variable itself.

So let's see how this looks.

Python of Hello.py, Enter.

What's your name?

David.

Enter.

OK, I've kind of over-corrected.

Now I've got two spaces for some reason.

Well, it turns out, and this is subtle, when

you pass multiple arguments to print, it automatically inserts a space for you.

This was not relevant earlier because I was

passing in one big argument to print all at once by using that + operator.

This time I'm passing in two because of the comma.

So if I don't want that extra space, I don't need to pass in one myself,

I can just do this and now notice, if I run this program again--

Python of Hello.py, type in my name David,

now it looks grammatically like I might want.

Now which of these approaches is better?

This approach uses a function print with two arguments--

Hello, comma and the name variable.

The previous version, recall, technically used one argument,

even though it looked a little curious.

It's one argument in the sense that the computer, just like mathematicians,

are going to do what's inside of parentheses first.

So if inside of parentheses you have this string of text--

hello, comma, and a space, which I need to add back.

Then you have a +, which means not addition, per se,

but concatenation-- to join the thing on the left and the thing on the right.

This ultimately becomes the English phrase--

Hello, comma, space, David.

And then what's being passed ultimately to the function

is technically something like this, but it's doing it all dynamically.

It's not me typing in David as I discreetly did earlier.

It's figuring out dynamically what that value is after concatenating Hello

with the value of name and then passing that ultimately

to print as the sole argument.

Let me pause here to see if there's any questions on numbers of arguments

now to functions.

AUDIENCE: Can we use a function many times

to solve a certain problem which we can encounter many times in our code?

DAVID MALAN: You can.

You can use a function many different times to solve some problem.

What we'll soon see, though, is if you find yourself

as the programmer solving a problem the same way again,

and again, and again, it turns out you'll

be able to make your own function so that you

don't have to keep reusing the basic ones that come with the language.

AUDIENCE: I was curious about the comma and the + sign.

So after + sign, can we give just one variable and after comma again we

give multiple variable status?

What is the difference?

DAVID MALAN: A good question.

So in the context of strings--

and I keep using that term.

"String" is a technical term in a programming language

and again, it means a sequence of text-- a character, a word, a whole paragraph,

even.

So the + operator is not just used, as we'll

see, for addition of numbers in Python, like we do on paper pencil,

but it also is used for concatenation of strings on the left and the right.

If you did want to combine not just two strings--

left and right-- but a third and a fourth,

you can absolutely keep using +, +, +, +, and chain them together

just like in math.

Eventually that's going to start to look a little ugly,

I dare say, especially if your line of code gets long.

So there's better ways that we'll actually soon see.

And a good question as well.

Well let me come back to the code here in question

and see if we can show you just a couple of other ways

to solve the same problem, along the way emphasizing that what we're technically

talking about here, yes, are strings, but there's even

a technical term for these strings in Python,

it's just STR, so to speak, S-T-R for short, for string.

As you may know if you programmed in other languages,

people who invent programming languages like to be very succinct, to the point,

so we tend to use fairly short phrases to describe things,

not necessarily full words.

So while you might say "string," technically

in Python what we're really talking about, these sequences of text,

are technically STRs.

This is an actual type of data in a program.

But we'll soon see that there's other types of data in programs as well.

In fact, let's see if we can't improve this in one other way.

I like the progress we've made by keeping everything on the same line--

Hello, David all on the same line.

What more though could we do in terms of solving this problem?

Well, it turns out that we didn't have to give up entirely

with using print twice.

Let me rewind a little bit and go back to that earlier version

where I wasn't really sure how to solve this problem so I was using print once

to print out just the Hello and the space and the comma.

And then I used print again to print name.

That's, strictly speaking, wasn't bad, but there was this visual side effect

that I didn't like.

It just looked ugly to have these two lines of text separate

from one another.

But there's another way to fix this.

Clearly it seems to be the case that the print function is automatically

outputting a blank line.

It's moving the cursor automatically for me to the next line

because that's why I'm seeing Hello on one line and David

on the next and then my prompt--

the dollar sign-- on the line below that.

So print seems to be presuming automatically

that you want it to move the cursor to the next line

after you pass it some argument.

But you can override that behavior.

Again, functions take arguments which influence their behavior,

you just have to know what those arguments are.

And it turns out that if we look at the documentation for Python's print

function, we can actually look up at this URL here--

docs.python.org is where all of Python's official documentation lies.

If I poke around, I can find my way to more specifically this URL

here where I can find all of the available functions in Python

and the documentation there for.

And if I go a little more precisely, I can even

find specific documentation for the print function itself.

And rather than pull that up in a browser,

I'm going to go ahead and highlight just one line from that same URL, which

is this, and this is easily the most cryptic thing we've seen yet,

but this is the official documentation for the print function.

And one of the best things you can do when learning a programming language

is, honestly, learn to read the documentation,

because truly, all of the answers to your questions will in some way

be there, even though, admittedly, it's not always obvious.

And I will say too, Python's documentation

isn't necessarily the easiest thing, especially for a first time or novice

programmer.

It too just takes practice, so try not to get overwhelmed if you're not

sure what you're looking at.

But let me walk you through this example.

This again is a line of text from Python's official documentation

for the print function.

What this indicates as follows is this--

the name of this function is, of course print.

Then there's a parenthesis over here and another close parenthesis way

over there.

Everything inside of those parentheses are

the arguments, the potential arguments, to the function.

However, when we're looking at these arguments

in the documentation like this, there's technically a different term

that we would use.

These are technically the parameters to the function.

So when you're talking about what you can pass to a function

and what those inputs are called, those are parameters.

When you actually use the function and pass

in values inside of those parentheses, those inputs,

those values are arguments.

So we're talking about the exact same thing-- parameters and arguments are

effectively the same thing, but the terms you

use from looking at the problem from different directions.

When we're looking at what the function can take versus what

you're actually passing into the function.

So what does this imply?

Well this syntax is pretty cryptic, but at the moment,

just know that an asterisk, a star, and then the word "objects"

means that the print function can take any number of objects.

You can pass in 0 strings of text, one string

like I did, two strings like I did, or, technically, infinitely many

if you really want, though that code is not going to look very good.

After that we see a comma, then we see another parameter here

called SEP, short for separator in English.

And notice the equal sign and the single quote, space, single quote.

So 'space' I don't know what that is yet,

but I think we've seen a hint about it.

Let's focus though for a moment on this--

the print function takes another parameter called end and the default

value of that parameter is apparently, based on this equal sign and these

quotes, /n.

And what is /n, if you'd like to chime in in the chat?

Anyone who's program before has probably seen this,

though if you've never programmed before, this might look quite cryptic.

Backslash n means new line, and it's a way textually

of indicating if and when you want the computer effectively

to move the cursor to the next line, create a new line of text.

And so technically, if we read into the documentation,

we'll see more detail on this.

The fact that there's a parameter called end in the documentation for the print

function, just means that by default this print function is going to end

every line with /n.

You don't literally see /n, you see a new line.

You see the cursor moving to the next line.

Now by that logic, let's move backwards.

SEP four separator-- the default value of separator

is apparently a single blank space.

Well where have we seen that?

Well recall in an earlier example when I passed in not just one but two

arguments to the print function.

Recall that they magically had a space between them.

In fact, they had that space + my own space

and that's why I deleted my space because at that point it was extra.

So this just means that when you pass multiple arguments to print, by default

they're going to be separated by a single space.

By default, when you pass arguments to print,

it's the whole thing is going to be ended with a new line.

Now just by knowing this, and let me literally

wave my hand at the rest of the documentation for another day,

there's more things that print can do, but we're

going to focus just on SEP and on END, let's see if we can't leverage this now

to solve that original problem.

The original problem was this, I don't like how Hello, David

is on two different lines.

Well that's happening again because print is automatically

printing out a new line, so let's tell it not to do that.

Let's tell it by passing a second argument to the first use of PRINT

to say END = quote, unquote--

not /n, which is the default automatically.

Let's make it quote, unquote nothing else.

Let's override the default value so there is no new line.

There's literally nothing there.

And let's see what happens.

Let me now go down to my terminal window and clear it.

And I'm going to run Python of Hello.py, Enter.

I'm going to type in my name, David and I

think now everything is going to stay on the same line because--

and it did-- this line here, 5, is going to print out Hello, comma, space,

but then nothing at the end of it because I

changed it to be quote, unquote.

The second line is going to print the name, David,

or whatever the human's name is, and it will move the cursor to the next line

because I didn't override the value of END there.

Just to see this more explicitly, if you do something cryptic like,

well I have no idea what's going on.

Let me just put in temporarily three question marks here.

We'll see the results of this too.

Let me go back down to my terminal window, run Python of hello. py,

what's your name?

David.

And now you see literally really ugly output,

but you see literally what's going on.

Hello, comma, space, then three question marks, END, that print statement,

and then you see D-A-V-I-D. So not a good outcome,

but it demonstrates just how much control we have here too.

And let me rewind further.

Recall that in our other version of this,

when I passed in Hello, comma and name, they were separated by a single space.

So Python of Hello.py, D-A-V-I-D, Enter.

That just worked.

Well what if we override the value of SEP for separator?

Instead of being one space, we could say something

like, question mark, question mark, question mark just to wrap

our minds around what's going on there.

Let me now do Python of Hello.py, D-A-V-I-D, Enter.

And you see two.

These two inputs--

Hello, comma and the name are now separated in an ugly way

by three question marks because I've overridden the default behavior of SEP.

And even though the documentation uses single quotes,

I've been in the habit of using double quotes.

In Python you can use either.

Strictly speaking, it doesn't matter, but you should be consistent

and I generally always use double quotes.

Python's documentation, though, always uses single quotes.

Questions now on these types of parameters?

And allow me to propose that we give these an official name.

Up until now, when we've been passing values to print,

those are called positional parameters-- positional in the sense

that the first thing you pass to print gets printed first.

The second thing you pass to print after a comma gets printed second.

And so forth.

But there's also these things we've now seen called named parameters.

Named SEP, separator, or END, E-N-D for the line ending.

Those are named parameters because one, they're optional

and you can pass them in at the end of your print statement,

but you can also use them by name.

AUDIENCE: This may be a weird question, but I was wondering,

what if someone wants to add actually quotation marks within the quotation

marks?

DAVID MALAN: Yeah, I like how you think.

This is what we would call a corner case, right?

Just when we've made-- right, this is all

sounding great, at least as programming goes, but, wait a minute,

what if you want to print a quote?

That's a really good question.

Well, let's see if we can't figure this out.

Suppose that I want to print out not just the user's name.

Let me simplify this further.

Let me go ahead and get rid of a lot of this,

and let me just say something like, Hello--

maybe I'm being a little sarcastic here.

Hello, (sarcastically) "friend."

You know, in that kind of tone.

Well this is not going to work, actually,

because you are trying to use quotes to be like "friend" in finger quotes,

but you're also trying to end the sentence.

And if I try running this, let's do this.

Python of Hello.py, you'll see that this is just invalid syntax.

Perhaps you forgot a comma.

And this is actually a bit annoying.

Sometimes the error messages you see are misleading.

Like the computer, the language, doesn't really

know what's going on so it gives its best

guess, but it's not necessarily correct.

But I can solve this problem in a couple of ways.

I can do this--

I can change my outermost quotes to single quotes.

Because recall a moment again, I said you

could use double quotes or single quotes so long as you're consistent.

So that's fine.

If you use single quotes on the outside, you

can then use double quotes on the inside and you'll see them literally.

So for instance, if I run Python of Hello.py, there we go.

Hello, "friend."

But there's another way.

If you insist on using double quotes as you might want to,

just to be consistent, you can also use that backslash character again.

We saw the /n a moment ago.

And that meant we don't want a literal n to be in the output,

we wanted a new line.

So the backslash actually represents what's called an escape character.

An escape character is one that you can't just

type necessarily once on your keyboard.

You need to express it with multiple characters.

So I can actually put backslashes in front of these inner double quotes

so that the computer realizes, Oh wait a minute,

those aren't quotes that finish or start the thought, they're literal quotes.

So now let me go back to my terminal window, run Python of Hello.py, Enter.

And now it's working as well.

So escaping is a general technique that allows us to do that too.

And if I may, let me rewind now on these examples

and go back to where we left off with my code,

I'm just undoing all of that, because I want to get back to the point

ultimately of specifying now a final way of solving this problem.

Well, it turns out that we have yet another way

we can solve this problem, which is perhaps the most frequently done

now or at least the most elegant when it comes to setting us up

for longer and longer uses of strings.

You can use a relatively new feature of Python that allows you to do this.

You can literally put, not the name of the variable like that in your string,

because we already saw this is wrong, right?

If you do this, you will literally see Hello, comma, name.

But what if I do this?

What if I put curly braces or curly brackets around the variable's name?

Notice VS Code is actually very subtly changing the color of it.

So VS Code knows something interesting is going on here.

Let me run this program, but I'm not done yet Python of Hello.py, Enter.

D-A-V-I-D, Enter.

OK, obviously not what I want, but I need to tell Python

that this is a special string.

This is what we're going to call a format string or an F string,

a relatively new feature of Python in the past few years

that tells Python to actually format stuff in the string in a special way.

And the symbol via what you do this is a little weird,

but this is what the world chose.

If you put an F at the beginning of the string,

right before the first quote mark, that's a clue to Python that, ooh,

this is a special string.

Let me format this in a special way for you.

Let me now rerun the program.

Python Hello.py, Enter.

D-A-V-I-D, Enter.

And now we see the goal this whole time--

Hello, David.

We don't start with this way, because I think if we did this the first way,

you'd be like, why are we doing this?

What are all these magical symbols?

But this is just yet another way to solve the same problem.

But let me propose that we consider now yet

other things we can do with strings.

And it turns out that even as we've been doing some relatively simple operations

here, we've generally been trusting that the user is going to cooperate

and that is to say that they're going to actually type in what we want them

to type.

Now just because they type a string, though, doesn't mean

it's going to look the way we want.

You and I, honestly, as humans are actually

in the habit-- on websites and apps-- of like accidentally hitting

the spacebar a lot, either at the beginning of our input

or at the end, maybe because the space bar tends to be so big.

It's pretty common to get accidental spaces

before or after some users' input.

You and I are definitely in the habit of not necessarily capitalizing words

like we should.

If we're sending text messages, we're probably being a little quick

and just sending everything in lowercase,

for instance, if that's your style, if your phone's not fixing it for you.

Maybe in a formal letter you would capitalize things properly.

But you and I as humans, we can't really be trusted to type things in a nice way

necessarily when using some piece of software,

be it an app or a website or something else.

But it turns out that strings themselves come

with a lot of built-in functionality.

You can see all of that in Python's own documentation here.

The string data type that we've been talking about

comes with a lot of functionality built in that

means that we can manipulate the user's input to do more than just

join it with something else like Hello, we can actually clean it up

or reformat it in a way that hopefully looks a little better for us.

So let me go back to my code here and let me just

demonstrate what might happen if a user doesn't cooperate.

If I go ahead here and run Python of Hello.py, Enter.

Let me just sloppily hit the spacebar a few too many times.

Why?

I just wasn't paying attention.

And now I'm going to type in my name D-A-V-I-D and, I don't know,

I hit the spacebar a couple more times.

Like it's kind of a mess.

It's all lowercase.

That's not going to necessarily look grammatically right.

It's got spaces here and here.

The program is going to print exactly that

and that looks really bad, at least if we're

prioritizing aesthetics and grammar.

Like, why are there so many spaces after the comma?

This is not a very nice way to greet your users.

But we can clean this up.

It turns out that built into strings, which, again, is this data type, so

to speak, this type of data in Python--

is the ability to actually do things to that string.

So let me do this, I can actually go ahead and do something like this--

name = name.strip.

And what does this do?

Remove white space from string.

And what do I mean by this?

Well, on the right-hand side, notice I've

written the variable name called Name.

I've then used a period or a dot, and then I

seem to be doing what's a function, right?

Any time we've seen a function thus far, we see the function's name-- print

or input, then we see a parenthesis, then another parenthesis,

and that's exactly what I see here.

But I'm using this function a little differently.

Technically, this function is-- in this context-- called a method.

And what do I mean by that?

Well, if Name is a string, a.k.a.

STR, well it turns out, according to the documentation,

there's a lot of functions that come with strings in Python.

And you can access that functionality by using

the name of a string like literally name here, then a period, then the name

of the, function and then an open parenthesis and a closed parenthesis.

Maybe some arguments inside of those parentheses, but in this case,

it doesn't need any arguments.

I just want to strip the space from the left and the space

from the right of the user's input.

But that's not enough.

I want to remember that I've stripped off

that white space on the left and the right,

so I'm going to use the equal sign again here.

And notice that just as before, this doesn't mean equality,

this means assignment from right to left.

So when this line of code here-- name.strip--

returns to me, a.k.a.

a return value, it will return the same thing

that the user typed in, but with no more white space to the left

or to the white [CHUCKLES] to the right.

So then the equal sign assignment is going

to copy that value from the right to the left,

thereby updating the value inside of my name variable.

So you can not only assign values to variables,

you can absolutely change the value of variables

by just using the assignment operator, the equal sign again,

and again, and again, and it will just keep copying from right

to left whatever the new value should be.

So now if I rerun this program, Python of Hello.py, Enter.

I have D-A-V-I-- oop, let's do it again.

Space, space, space, space, space. d-a-v-i-d in all lowercase, space,

space, Enter.

It's better.

It hasn't fixed my capitalization, so I'm still

being a little sloppy with the first D, But it has stripped off

all of that extra space.

Super minor detail, right?

Like this isn't all that exciting, but it just

speaks to the power of what you can do with just a single line of code.

Now what else can I do here?

Well I could capitalize the user's input.

Let me go ahead and try this.

It turns out that I could also do this-- name.capitalize.

So let me go ahead and capitalize user's name.

And again, I'm making comments and there's

no one right way to write the comments.

I'm just using some short English phrases here

to remind myself of what I'm doing.

What's now going on here?

Well let me go ahead and run Python of Hello.py, Enter.

Space, space, space, space, space, d-a-v-i-d, space, space, Enter.

OK.

Now it's looking prettier, right?

No matter how the user typed in their name, even a little sloppily,

I'm now fixing that.

But let's try something.

I'm getting a little curious here.

How about this?

Space, space, space, space, space, d-a-v-i-d space m-a-l-a-n--

I'll use my last name now--

Enter.

OK, so ironically, "capitalize" is not really capitalizing everything we want.

It's clearly capitalizing what?

Just the very first letter.

So it turns out that, again, there's other functions in Python

that come with strings.

And if we poke around the documentation, scrolling through a URL like that,

I bet we'll find another solution, one of which is actually this.

Let's actually change this to title.

There's yet another function that comes with strings

called Title that do title-based capitalization, just

like a book or a person's name, capitalizing

the first letter of each word.

And this is just going to do a little more work for us.

So let's go ahead and run this.

And as an aside, I'm kind of tired now at this point

of typing Python, Python, Python all the time.

It turns out that when using a command line interface like this,

you can actually go back through all of your old commands.

What I just did a moment ago was I hit the up arrow.

That immediately goes back through my history of all of the commands

I've ever typed.

So this is just a faster way now for me to repeat myself

than typing everything manually.

Let me go ahead and hit Enter, space, space, space, space, space,

d-a-v-i-d m-a-l-a-n space, space, all lowercase, Enter.

Now it's looking better.

Now I've capitalized things and cleaned things up.

But what about my code?

I've got like eight lines of code now, four of which

are comments, four of which are actual code.

Do I really need this much?

Well, not necessarily.

Watch what I can also do in Python.

Let me not bother capitalizing the user's name separately.

Let me say this--

and capitalize user's name.

I can chain these functions together.

I can add Title to the end of this.

And now what's happening?

Well again, with a line of code like this,

you first focus on what's to the right of the equal sign,

then we'll get to the left of the equal sign.

What's on the right of the equals sign?

This line here.

Well what does this mean?

Get the value of the name variable, like D-A-V-I-D space, M-A-L-A-N.

Then strip off the white space on the left and the right.

That is going to return a value.

It's going to return D-A-V-I-D space M-A-L-A-N without any white space

to the left or right.

What do you want to do with that return value?

You want Python to title case it, that is

go through every word in that resulting string

and fix the first letter of the first word, the first letter

of the second word, and so forth.

And then-- now we can finish our thought--

copy the whole thing from right to left into that same name variable.

And you know what?

I can take this even one step further.

Why don't we go ahead and do this if we want.

Let me get rid of all of that, and let me just do strip and title

all on that first line.

And now we've gone from like eight lines of code to four.

It's a lot tighter.

It's a lot neater.

And even though reasonable people might disagree,

it's arguably better because it's just easier to read.

Fewer lines of code, fewer opportunities for mistakes,

it just allows me to move on with my next problem to solve.

All right, let me pause here and see if there's any questions on these methods.

A method is a function that's built in to a type of value,

like these functions are.

Or on F strings which we saw a moment ago.

AUDIENCE: Yes, hi.

Thanks, David.

So is there a way to remove the spaces between the spaces

that I might have added?

DAVID MALAN: Short answer, no.

If you read the documentation at that same URL earlier,

you'll see that strip removes from the left and the right, but not in between.

In fact, there's two other functions that come with strings.

One is called L strip, the other is called R strip,

that allow you to do one or the other.

If we want to start getting rid of space in the middle,

we're going to have to do a different trick altogether.

AUDIENCE: How many functions can be combine like this dot strip, dot title,

all combined.

So how many we can combine?

DAVID MALAN: Yeah, a really good question.

Technically, as many as you want, but at some point

your code is going to start to look really, really bad, right?

Because the line of code is going to get really, really long.

It's eventually going to maybe wrap around again and again.

So at some point, you just kind of say like, uh-uh, that's too many.

And you start breaking it up into multiple lines like I did.

Maybe reassigning the value to the variable as needed.

And this is actually a good question.

If I can pivot, [INAUDIBLE] off your question, I mean, what do people think?

If we could go ahead and put everyone's hands down for a moment.

Let me ask this--

is the way I've done this now, with strip and title and input

all in the same line--

better than my previous approach?

In Zoom you can use the Yes icon or the No icon.

If you think this version is better, say Yes.

If you think this previous version was better, for instance,

this one here where we had everything broken out, say No.

And then we'll see why in just a moment.

I proposed earlier that reasonable people can disagree

and that's absolutely the case.

Doing it one way or the other isn't necessarily best,

at least if you can justify it.

Let me go back to the most recent version here.

All right so we're seeing a lot of Yeses and a lot of Nos.

Why don't we go ahead and call on one of the Yeses, if we could.

Someone who's voting Yes, why do you think the current version of this code

is indeed better than the previous longer version of the code?

AUDIENCE: I think it's more readable.

So I can say, this is the name variable.

It gets some input and then remove the space and give it a title.

And there you go.

You have a hello, name.

DAVID MALAN: Yeah, I think that's pretty reasonable.

It's very readable, at least if you're in the habit,

as you are in English, of reading left to right.

It just kind of flows very naturally as a result.

The lines is not really that long.

It's certainly fitting nicely onto the screen.

So I think that's a good argument.

How about a counterpoint, though?

Someone who voted No, if we could call on someone who thinks this is worse.

AUDIENCE: Because it's not readable at all.

It seems like it's a very long line.

So I think it's better to separate.

DAVID MALAN: Yeah.

I think that's persuasive too, right?

It's getting a little longer.

And even though my sentence here-- what's your name-- is relatively short,

you could imagine that this could get even uglier quickly

if I were asking a longer question of the user, that's

going to make this line of code even longer and therefore less readable.

It might be less obvious to me or my colleagues that I am calling strip

or that I am calling title.

It might be kind of an unexpected surprise.

So I think that's reasonable too.

In short, there is no right answer here.

And in fact, part of the process of getting better at programming

is getting your own sense of style or working for a company

where they might prescribe which way is better than the other

because they just want everyone doing the same thing,

even though reasonable people might disagree.

Ultimately though, so long as you have what's

a pretty good argument in favor of one way or the other, like ultimately,

that's what's important.

If you're just doing things because you don't really know which one is better,

that's not great.

But if and when you start to acquire opinions

and if your boss, if your teacher, if your colleague, your friend,

can challenge you and say, wait, why did you do it like this?

They might not agree with you, but at least have an answer,

and that should be sufficiently persuasive in general.

Now, strings come with a whole bunch of other methods

as well, among which is one called split, which can, as the name suggests,

split a string into multiple smaller substrings, so to speak.

For instance, if the human here is in the habit

of typing in their first name, then a space, and then their last name,

and you want to go ahead and greet them only by first name,

well we could actually leverage that single space

between the first name and last name and split that string

into two smaller substrings.

How can we do this?

Well, let me go ahead and in between these lines

proactively comment that we're about to split user's

name into first name and last name.

And then let's go ahead and take that name variable, which currently contains

something like, presumably, David, space, Malan,

and let me go ahead and call split and pass in as the argument

to split a single white space, thereby indicating that I indeed

want to split on that character

Now it turns out split is going to return a sequence of values, ideally

a first name and then a last name.

And we can actually, in Python, assign both of those values

from that sequence at once to some variables.

For instance, first comma last equals, and that's

going to have the effect from right to left

of putting the first such value in the first variable, the second such value

in the second variable.

So now on my last line of code, I can go in and say hello

not to the full name, something like David Malan,

I can just say Hello, first.

All right, let's go ahead and clear my terminal window, run Python of Hello.py

and hit Enter.

I won't bother with any leading white space this time, but let me go ahead

and type in David space Malan, and crossing my fingers as usual, Hello,

David is what we now see.

All right, so we've seen so much, so many examples thus far involving

strings, but certainly programs and programming languages can

manipulate other types of data as well.

Let's go ahead and transition then to another very common type of data

in Python, in programming more generally, namely

integers, otherwise known in Python as INT--

I-N-T. So just as STR, S-T-R is short for string, so is INT in Python

short for integer.

Well, what's an integer?

Well just like in math it's a number like negative 2,

negative 1, 0, 1, 2, and all the way toward negative infinity, all

the way toward positive infinity.

But there's no decimal point in an integer.

It's just a number like negative 2, negative 1, 0, 1, and 2 onward.

That's an INT.

Of course, in the world of mathematics, there's lots of symbols that we use.

And we've seen + before, although we used it for a different purpose.

But Python supports these symbols and more.

And Python allows you to add numbers together +,

subtract numbers, multiply numbers, divide numbers.

And the only one here that might look a little strange to people or unfamiliar

is this percent sign, but it doesn't mean percent in this context.

If you use a single percent sign in a Python program,

that's actually the so-called modulo operator,

the operator that allows you to take the remainder after dividing

one number by another.

So we'll see examples of that before long, but the first four of these

are perhaps quite, quite familiar.

Well it turns out that in Python you cannot necessarily--

you don't necessarily have to keep writing code in a file like Hello.py

and then running it in a terminal window.

One of the features that many people like about Python

is that it supports this so-called interactive mode.

Like you can start writing Python code and immediately execute

each of those lines interactively, especially

if you don't care about saving all of your lines of code.

You just want to execute code and get back some answers.

So for instance, let me go back to VS Code here and let me close Hello.py

and let me click on the little triangle over here in my terminal window just

to make it much bigger just temporarily for a moment.

So I'm not creating any .py file now, I'm just going to run Python by itself

at my prompt.

And you'll see when I do this, I get some cryptic looking output

and the date and time at which the program was last updated, and so forth.

But I ultimately get three triple brackets like this.

Is the interactive mode for Python.

So I'm running the Python interpreter and any time I type a line of code

in the interpreter, it's going to execute it immediately.

I don't have to keep running Python again and again.

It's as though in the human world, if you

were standing next to a human who speaks some other language,

and you're just having a conversation with them back and forth,

it's all happening-- the translation-- immediately.

So what might I do in interactive mode?

Well I could do something like 1 + 1, Enter.

That's actually code, all right?

You might not think of it as code, but if a bit of arithmetic and you

know numbers, and you know +, that's valid Python code.

And you can use Python really as a fancy calculator.

But I could do other things too.

If I want to print to myself Hello, world,

I can also print out that line of code there too--

Hello, world.

So it's interactive in the sense that the moment

you execute a line of code, boom, you see the result.

We're generally not going to do that because at least when

teaching the language, we tend to want to do things incrementally

and we want you to be able to see where it is we came from.

And we want to be able to try things again and again, especially

if we make mistakes.

But know that this is indeed a feature of Python,

this so-called interactive mode.

But let's focus for a moment now, not just on that interactivity,

but really on the fact that Python apparently

supports integers and mathematics and some of those basic operations.

And let's see if we can't make maybe our own little calculator.

So let me go ahead and open up VS Code again,

and I'm going to shrink down my terminal window,

and I'm going to create a new file called Calculator.py.

So to do that recall, I can type Code down here,

and the name of the file I want to create, .py, Enter.

That gives me a new tab up top.

So I have already closed Hello.py.

I'm now in Calculator.py.

And let's just make a simple calculator that does some addition for me.

But I'm going to do it in a file so that we can iterate on this

and make changes for better or for worse over time.

Let me go ahead and first declare a couple of variables.

I'm going to do the mathematical thing of calling my first variable x,

my second variable y, and then I'm going to give myself

a third variable z = x + y.

And then I'm going to go ahead and print out z.

Now this program, admittedly not very exciting or interesting,

in fact, it's a little less interesting than printing stuff on the screen

like before with strings, but we'll build on this

and see what other features exist in Python that we can leverage.

So hopefully, if Python knows it's math as well as I do,

when I run Python of Calculator.py, I should see hopefully that 1 + 2

= indeed 3.

All right, so not that surprising and not that interesting.

And honestly this isn't the most useful program

because it's always going to calculate 1 + 2 = 3.

Let's at least make this program, say, a little more interactive, right?

We already know from previous examples how we can get input from the user.

Let's bring back that input function.

And let's do this, let me go ahead now and at the top of my code,

let's change x to not be the number 1 always,

let's change it to be whatever the return

value is of asking the user for x.

And I can use any English or human language I want here.

I'm going to say, what's x?

Just like I asked before, what's your name?

And I'm going to do the same thing for y.

I'm going to use Input again, but this time change the question to be,

what's y?

All right, at this point, I think I'm going to leave the rest of the code

the same.

Z = x + y And then print z but what's nice

now is that I think I have a nice interactive calculator.

Right, now it's not going to do 1 + 2 all the time.

It's going to do whatever the user types + whatever the user types.

So let's try this.

Let me go ahead and run that program.

All right, let's do it.

1 is going to be x, 2 is going to be y, and of course, everyone in agreement, 1

+ 2 = thr--

huh.

What's going on there?

Either your math class misled you or I have misled you.

Why don't we call on someone here to see if you

can't help us reason through what the bug is, what's the mistake?

Anjali, if I'm saying it right.

AUDIENCE: I think the issue is that it's concatenating strings because you

use the + operator instead of adding.

DAVID MALAN: Perfect.

So perfect intuition.

We've seen that + is used a little differently in the context of strings

because it concatenates, that is, it joins the two strings,

and that seems to Indeed be what's happening here,

even though the user typed a number.

But the interesting thing here is that, when

you get user input, because they're using a keyboard on their Mac

or PC or their phone, it is always going to be text.

It might look like a number, but by default,

it's coming from the keyboard as a string--

that is, as text.

And so, how do we go about resolving this if ultimately we

don't want to treat those inputs as strings,

we want to treat them as actual numbers?

Well we need another function and it turns out in Python

that you can convert sometimes from one type of data to another type of data,

for instance, from string to INT by doing something like this.

Let me go back into my code and let me change

x before adding it to y to be whatever the integer version of x

is, + whatever the integer version of y is.

So it turns out that INT is not only a type of data in Python,

it's also a function, and it's a function

that if you pass in an input, like a string,

so long as that string looks like a number like 1 or like 2,

it will convert it to an actual number that you can perform mathematics

on instead.

So if I now go back to my terminal window and run Python--

and let me show you another trick.

"Calculator" is kind of a long word.

It's a little tedious to type.

Notice what I can do in my terminal window, in a command line

interface in general.

If I start typing C-A-L for calculator, I can actually

hit Tab to finish my thought.

So auto-complete is possible in a terminal window like this.

Type the first letter or few letters and then, boom, with Tab,

it'll finish your thought for you.

Or you can go back in your history like I did with the Up and Down Arrows.

Let me go ahead and execute this.

What's x?

1.

What's x?

2.

And there we go.

Now we have a general purpose calculator that's going to support

not just the addition of 1 and 2, but now any two

integers that the user types.

And let me now improve this, right?

We've seen how we can make improvements to code

and I don't know if it's going to necessarily be better,

but let's try this.

Do I really need the z variable?

It's worth noting that I'm creating a variable called z,

and then I'm immediately using it on the next line of code.

Now that's not that compelling, because if you're creating a variable

and then immediately using it, but never again using it,

did you really need to take the time to introduce another symbol

and another variable just to use it once and only once?

Well, maybe not.

Maybe we don't really need z in this way.

Maybe I should go and do something like this.

Maybe I should get rid of z here.

Maybe I should change this to be INT up here, change

this to be INT up here, doing something that's pretty interesting now.

Even though it's a bit of new syntax, notice that you

can nest functions, so to speak.

You can put one function call that is the use of a function

inside of the use of another function so that the return

value of the inner function becomes the argument to

or the input to the outer function.

Just like in math, if you have parentheses, parentheses, parentheses,

your teacher probably taught you to focus

on what's inside the innermost parentheses first

and then work your way out.

Same thing with programming.

That's what's Python is going to do.

It's going to look at what's inside of the parentheses first,

it's going to get the answer, and then it's

going to pass the return value to the outermost function.

So what happens on line 1 now is that the input function gets called first,

then the result of that, quote unquote one

becomes the input to the INT function, and same on line 2.

The output of what's y becomes the input to this INT function.

And now there is no z.

I could just do print x + y.

And because I've taken the time to convert each of those strings

to an integer, I think we're OK.

So let me try this, Python of Calculator.py, Enter.

1 and 2, and we're still getting 3.

Not 12, or not 12, 1,2, we're indeed getting 3.

And we've additionally gotten rid of the variable

because we didn't necessarily need it, it seems,

especially if only using it once.

Well here too, let me put everyone's hands down for just a moment

and let me ask as before--

this version now which uses INT around the invocations of input,

and does not use z, is this better than the previous version?

If you want to vote Yes, go ahead.

Or if you prefer the old way, vote No.

The old way, I'll undo all of this as we vote, instead looked like this.

All right, and let me go back to now the newest version.

Let's take a hand of the Yeses, someone who

thinks this latest version is better.

AUDIENCE: I think this way is better because it allows us to immediately see

what the x and y variables are with integers

and so we know what to expect from them.

And also the print argument is more intuitive.

We avoid too much clutter in the code.

DAVID MALAN: I think those are all good reasons.

It's nice and succinct.

The lines of code are not very long.

I don't need to know what z is because it doesn't exist.

It just sees print x + y.

I like that.

But someone who prefers the older way where we did have z

and we more explicitly passed individual variables to the INT function.

AUDIENCE: Yeah, hi.

I think that the earlier version is better because when--

I mean, if user input something else other than,

let's say, I mean, let's say, they type 1 and 2, like,

so it will be easier to debug.

DAVID MALAN: This version or-- this version here or the old version?

AUDIENCE: The old version.

DAVID MALAN: OK, that's fair.

And in fact, I'm being very careful today, as best I can, not to mess up.

I have thus far only inputted integers when I'm expecting integers.

And Raoul's actually pointing to something

we'll come back to in the coming weeks, how do we actually handle errors?

What if the user doesn't type in the number 1 or the number 2, or a number

at all?

What if they type in a word like cat, C-A-T?

That's not a number, and I bet I can't convert it to an integer.

But for today, I'm not going to focus on that.

I'm just going to hope that the user cooperates.

But that's not going to be the case.

And so perhaps one way would set us up for more success

when it comes to handling those errors.

Now for today's purposes, which is better?

Eh, I mean, I like both.

And I think both of you made very valid arguments in there too,

so long as you have a justification that feels pretty reasonable.

I mean, that's what ultimately matters.

But acquiring, again, a sense of the trade-offs here.

Well, is this way better?

If so, why or why not?

Just understanding what those trade-offs are.

But generally speaking, prioritizing readability is a very good thing.

Making your code readable for someone else is a very good thing

and very good for you too so that when you wake up the next morning,

or you come back the next week, or the next year,

you too can read your own code without having to waste

time trying to remember what you did.

And simplicity tends to be a good thing too-- keeping your code simple.

So as you get more comfortable with programming,

you might be tempted to try to combine an entire program into one long line.

For instance, let me do just that.

Technically speaking, we don't really need x in a variable.

We don't really need y in a variable.

We could also do this--

I could just get rid of x and y altogether.

I could then now eliminate that and make it just one line of code.

OK, so on some sense, you might be inclined to think, wow,

that's really nice.

You made it one simple line of code.

I would argue this actually isn't that simple.

Now I think I'm starting to nest too many things.

I have to think about print and INT and input.

I then have to notice that, OK, I've opened two parentheses,

I've closed two of them.

There's a +.

You're making me think too much and any time you make me think,

you're wasting time.

And any time you complicate the look of the code like this,

you're just going to increase the probability of mistakes

and tactical mistakes or logical errors in your code.

So if all the things we've done, this is the only one that I would argue,

yes, it's one line and it's nice and compact.

It's just not readable enough.

I would shy away from doing this, especially since two of those function

calls are getting input from the user.

But there too, reasonable people might disagree,

but that's the kind of visceral reaction you should have sometimes when

code starts getting a little too complicated, a little too clever,

perhaps, for its own good.

All right, well it's not just integers we have access to.

Let me propose that we transition from integers to one more data type here,

namely a float.

So again, a string is a sequence of text.

An INT is an integer like negative 1, 0 and 1.

A float is a number with a decimal point, properly called a floating point

value.

And you can think of the floating point as being

the decimal that might be over here or over here with some number of digits

to the left or the right.

Mathematically, it's a real number, a number that has a decimal point in it.

So that's a third type of data that Python supports.

Right now our calculator is somewhat naively

assuming that the user is only going to type in integers,

but if I want to support floating point values too,

I think I can just make a couple of tweaks.

So I'm going to go back to VS Code here.

And instead of just converting the user's input x and y to

integers on line 1 and 2, let's just make a simple change.

Let's actually convert it to a float on the first line

and a float on the second line here.

Now I think, if I go down to my terminal window and run Python of Calculator.py,

let's type in a number like 1.2 with a decimal point

and 3.4 with a decimal point, and there we go.

We have 4.6 as the final answer.

So that wouldn't have worked before if I was only expecting integers

from the user, but now that I'm expecting floating point values

and accommodating it, I can actually now do floating point arithmetic as well.

But suppose that I don't really want the final answer

to be a floating point value like 4.6.

I would be happy if we just round to the nearest integer.

So I want to support the user typing in floating point

values with decimal points, but at the end of the day,

I just want to round the result to the nearest possible integer, for instance.

Well it turns out that here too Python comes with some functionality built in.

And in fact, if we return to this URL from earlier

wherein all of the Python built-in functions are listed,

there's one called "round" which does exactly

as we would expect, it takes as input a number

and then rounds it for us, for instance, to the nearest integer.

But if we look a little closer at that documentation as we can here--

I'll provide an excerpt--

this is what the function looks like in the documentation.

And recall that earlier we looked at the documentation for print

and this is similar in spirit that this shows us not just

the name of the function but its available parameters, that is,

inputs that we can provide when using this function.

But this is a little cryptic too, just like Print was,

and it adds some syntax.

So let's see.

The name of this function here is of course Round

and its first argument is a number.

Notice this time there's no star, there's no star objects

like there was for print.

The Round function takes just one number as its first argument, period.

That's its positional parameter.

But notice this syntax.

And this is a convention in programming or technology

more generally, generally speaking, when you see square brackets

and documentation like this, this means that you're

about to see something optional.

And so what this means is that if you want

to specify more precisely the number of digits that you want the round function

to round to, you can specify it here by adding a comma and then that number.

So if we read the documentation, if you don't specify a number of digits,

you just specify the number to round, it rounds to the nearest integer.

But suppose you want around to the tenths place, or the hundredths place

that is one or two digits after the decimal point,

you could additionally pass in comma 1 or comma 2 to be more precise.

So that's what the documentation, there is saying.

Let's see if we can't then translate this to some actual code for us.

So if I go back now to VS Code and I consider

that I want to go ahead and round x and y, I can do this in a couple of ways.

I could do round x + y, but you know, I'd

actually kind of prefer to break this now out into two lines.

I don't have to, and reasonable people here might disagree,

but I'd like to revert to a scenario where I'm printing z,

so that I can just a little more clearly to myself, to others,

say z = the rounded result of x + y.

It's not necessarily the better way to do it,

but I'm a little more comfortable with breaking out my thoughts one at a time,

especially if I want to start commenting each of these chunks of code.

All right, let me go down to my terminal window

now and run Python of Calculator.py.

What's x?

Let's do 1.2 again.

Then let's do 3.4.

And now it was previously 4.6, but now it's

been rounded up to the nearest integer, which of course is going to be 5.

All right, what if I wanted to change this a little further?

What if I wanted to support maybe really big numbers,

big numbers, irrespective of rounding, let's just do something like this.

Let me go ahead and run Python of Calculator.py

again and let me just add 999 + 1.

And notice, I don't have to type decimal points,

even though I'm converting to float, my program

will just allow me to type decimal points, but I don't need to oblige.

The answer of course here should be, and is in fact, 1,000,

whether or not we round.

So that's just arithmetic with integers here.

But in the US, we tend to format long numbers by putting commas

after or before every triple of digits.

Other countries flip it and they use periods and commas instead.

That's a system setting.

You can change that on your own Mac or PC

or device for Python or any language, but for me, I'm

using the US approach here, which is periods for decimal points

and commas for separators.

What if I wanted this to be outputted as 1,000?

Just to make it a little more clear that it's 1,000 and not something like 100.

That's even more useful when it's like one million--

1,000,000.

Wouldn't it be nice if we could automatically

output those numbers as well?

Well, it turns out that we can.

There is a way using Python to actually specify that we

want to include commas like this.

And here we have an opportunity to bring back our old friend, the F string.

First, let me do something that's not that productive.

First let me do this.

Let me print out the value of z, but wait a minute.

I can't just say "z" because that's literally going to print z

on the screen.

So let me wrap it with those curly braces like I did before,

but that too was not enough.

I literally needed to add an F at the beginning of my string

to tell Python that this is an F string, a format string.

That now is going to print out, not very interestingly, just the value of z

itself.

So I'm going to great lengths just to print z when really I could have just

passed z as the sole argument.

But just to ensure that I haven't broken it, let's do this again.

999 + 1, Enter.

OK, it's still 1,000.

So I didn't make anything worse.

But notice-- and this syntax is unfortunately a bit cryptic--

notice that I can actually do this.

I can put a colon after the z and I can put a comma thereafter.

This looks very cryptic, admittedly, and even I

have to constantly look things like this up in the documentation

to remember the syntax.

But here, let me run it again.

Python of Calculator.py, 999 1 and now notice

that the number has been automatically formatted for me.

If I were in a different country or locale,

I could absolutely override this to use periods instead of commas or vise

versa.

But in this case here, it's just happening for me automatically.

So there too we see a hint of what it means to really format a string.

There's even more power--

more powerful capabilities built into that.

All right, let me pause here to see if there's

any questions now on floats, on rounding, or on this use of F strings.

AUDIENCE: Yes, so I have a question.

So when using floats, is there like a cap

to how many decimal points it can have?

DAVID MALAN: A really good question.

So floats, yes, and this is a problem we'll revisit before long.

Floats cannot represent numbers infinitely precisely.

In a nutshell, because computers only have so much memory.

They only have a finite amount of memory.

You and I only have a finite amount of hardware inside of the computer,

so at some point, they're going to have to round.

Right now I'm rounding automatically.

Effectively computers will eventually have to do that for us,

but we'll see that as a fundamental problem before long.

Allow me to turn back just for a few final examples on float

before we introduce a few final examples that allow us not just

to use functions, but to make our own.

Let me propose that we also try our hands at a bit of division here.

Let me propose that we modify this calculator now

to still take a couple of floats, but let's now just do

something a little simpler than--

a little different from this, just doing x divided by y.

And let me go ahead and get rid of my format string

and just keep it simple for now, printing out z instead.

And what are we going to see here?

Well just some simple division.

So Python of Calculator.py, let's do something like 2 divided by 3,

and of course I get 0.66666.

And to Ethan's question a moment ago, it does seem to be finite.

It's not rounding in a weird way here, but I only seem to see so many digits.

That's an inevitability of using a float in this way.

By contrast, just so you know, integers nowadays in Python

can be as big as you want them to be.

Unlike other languages, there is no upper bound

on how big an INT can be now in Python, but there

is a bound on just how precise a floating point value can be.

All right, now that I've got some simple division working here,

let's go ahead and round this.

It would be nice to round this really long number 0.6666666 and so forth

to maybe just two decimal places.

We've seen how to do this with round, though, at least in its documentation.

Let's just round this not to the nearest INT,

by passing in just x divided by y, which is one argument, once the math is

done inside of the parentheses, I don't want to pass in just one argument.

I want to pass in two so that I can specify n digits,

number of digits, which you'll recall was the second parameter for round.

Let me go ahead and run Python of Calculator.py.

I'll do the same thing--

2 and then 3, 0.67.

So here too we see a way of rounding now, not just to a nearest integer,

but to a nearest number of digits.

But there's another way to do this here.

And in fact, this evokes our F string example again.

Let me go ahead and change this.

Suppose that you didn't remember the round function or, for some reason,

you didn't want to use it.

You instead want to just use a format string.

Well, let's go there.

Let me do "z" but let me surround it with those curly braces.

Let me add the F at the beginning, and again, this is not interesting yet.

This is just going to print out z, but I'm adding a lot more complexity

to turn it into an F string.

But notice I can do something else after my variable name, after the colon.

If this were going to be a big integer, I

might want to use a comma like before to separate each triple of numbers

with commas.

But I don't.

I'm going to use a different sequence of characters.

I'm going to say 0.2F and this too is one of these very cryptic things I have

to constantly look up because I forget if I don't use it that often.

So don't be intimidated if this looks especially weird, but this is,

according to the documentation, the way you

specify using an F string how many digits you want to print.

So let me run this version of the calculator.

Type in 2 and then 3, we get the exact same thing.

But again, this is just consistent with my claim

that in programming we can so very often solve the same problem

in multiple ways.

This is just now the F string approach to that very same problem.

All right, which one is better?

It depends.

In this case, they're pretty equivalent.

You can imagine, though, it being useful to use a function sometimes

so that you can pass in an argument like n digits as that second argument,

or you can imagine just deciding in advance that you want 0.2

and then writing it like this.

Let's transition now from focusing on strings and on integers and on

floats to focusing now on functions themselves.

We began today by focusing on how you can

use functions that come with Python.

But wouldn't it be nice if you could invent your own functions,

especially if, to our point earlier, you find

yourself solving the same kind of problem again and again?

It's nice that Python comes with the print function

because it's really useful to be able to print things on the screen,

but wouldn't it be nice if you could print specific things on the screen

by just calling your own function?

Well let me propose that we do this.

Let me go back to VS Code here and let me propose that we go back to Hello.py.

I'm going to reopen Hello.py where we left it before

and I'm going to go ahead now and propose

that we consider how we can start improving this further by making

our own function.

I have written so many programs today that just say Hello

and each time I'm using print.

But wouldn't it have been nice if, from the beginning of today,

we could just call a function called Hello that just says Hello for us?

Now the authors of Python years ago didn't

think that we need a special function just to say Hello,

but I would like that to exist.

I'm saying Hello so many times, I just want

to be able to call a function Hello.

So I'm going to start from scratch here.

I'm going to delete all of my code from earlier

and I'm going to pretend for the moment that a function called Hello exists.

And I'm going to do just as I did before.

I'm going to get the user's name with the input function,

asking what's your name, question mark.

And now I'm going to call a function Hello

and then I'm going to print out the user's name.

Now I will admit, Hello doesn't exist, so bad things are about to happen,

but let's see what.

Let me go down to my terminal window.

Let me run Python of Hello.py.

I think the first line is going to be OK because that worked before.

And indeed, it's prompting me for my name.

So let me type in David.

The second line of code is apparently calling

a function that looks like it's called Hello, because why is it a function?

It has a parenthesis and a closed parenthesis immediately after it.

And that's what every function we've used has looked like.

But Python is not going to recognize this one.

When I hit Enter now, I get a name error.

Name "Hello" is not defined, did you mean Help?

I didn't, although it's opportune that's what I need at this point is some help,

but I am encountering this error because why?

The function just doesn't exist.

So how do I make this function exist?

Well I need to create it myself using this key word, DEF, DEF for define.

So here too, just as STR is short for string and INT is short for integer,

DEF is short for define.

If and when you want to define, create, invent your own functions,

you can do so using now this keyword in Python.

So let me go back to my code here and let me propose that we define this

perhaps in this way.

At the very top of my file, I'm going to first take

a moment to define a function called Hello

using DEF Hello, open parenthesis, close parenthesis, colon.

What this means now is that Python is going to treat every line of code

that I indent underneath this one as the meaning of this new function, Hello.

So DEF is important as is the space.

I get to choose the name of the function and I'm choosing to call it Hello.

The parentheses with nothing inside means that this function at the moment

is not going to take any inputs, no arguments there too.

The colon means, stay tuned for some indentation.

Everything that's indented beneath this line of code

is going to be part of this function.

It's going to be a super short function-- one line of code--

it's just going to print out "Hello."

But now on lines 1 and 2, I have invented my own function Hello.

Notice these dots that have now magically appeared here.

This is just a setting of my text editor,

VS Code in this case, that's just making super explicit to me that I've

hit the space bar four times, or equivalently the Tab key once,

which is converted automatically to four spaces.

Generally speaking, I'm going to need to make sure that all of my indented code

lines up now so that Python knows that it's all part of the same thing.

But it's easy in this case because it's just a single line.

But now, thanks to lines 1 and 2, the function Hello

will absolutely exist when I'm ready to use it on line 6.

So let me go down to my terminal window and run Python of Hello.py, Enter.

Here comes my name again.

And now when I hit Enter, I now see Hello, David.

All right, we've kind of regressed though, right?

This is not nearly as pretty as it once was.

I think we can probably do better than this by improving things further.

Why don't we consider, though, how we might

say parameterize this same function?

That is to say, can we customize Hello to maybe take the user's name as input

so that we can say, not only Hello, but the person's name all on one line, all

in one breath?

Well I think we can do this.

Let me propose that we do this as follows.

Let me go ahead and up in my code, let me inside of these parentheses,

let me come up with my own parameter name.

I have complete choice here and I'm going

to say that the name of my parameter will be the word To.

Why?

Because I want my function to sound like the verb it represents--

Hello.

But who do you want to say Hello to?

Well I'm going to call my parameter for this function

To, just because in English, it kind of sounds nice to me.

Hello To, who do you want to say Hello to?

That's why I'm calling this parameter To instead of something

simpler like x or y or z.

All right, well what do I want to do with the word To?

Well I can do a couple of different things.

We've seen like so many different ways to implement Hello.

Let me just add a comma there for grammar's sake.

And then let me put the word To after that as the second argument

to the function Hello.

There's other ways we can do this and we've seen so many,

but this one looks a little clear to me, I'll say.

What's going to happen next?

Well I don't think I need this extra print line here.

I think what I'm going to do is this, I'm going to go ahead here

and print out not the person's name manually, I'm going to instead say,

Hello parentheses name.

So what am I now doing?

On lines 1 and 2 I'm defining my very own function called Hello,

but this time that function has been designed

to take a parameter, a single parameter, as input.

And I'm using the value of that parameter which

I called To to plug into print so that I see not only

Hello but also that person's name.

What am I doing on line 5?

Same as always, I'm just getting the user's name.

Line 6, I'm not only calling Hello, I'm passing

as input the name variable as an argument

so that that's what gets passed into Hello.

And what's happening here is essentially this--

even though the variable is called Name here, when the function

itself is called, the computer assumes that same value is now called To.

So Name is essentially copied to another variable called

To so that in the context of Hello, I can say Hello to that variable instead.

And we'll see in a moment what happens if we don't keep those straight.

Let me go ahead and run Python of Hello.py, Enter.

What's your name?

And now I'm crossing my fingers.

Enter.

There we go.

We're back in business, but now I have my own custom function called Hello

that's allowing me to say Hello to a specific person.

And here's where now things can get really fancy.

What if you wanted your Hello function to say Hello to someone specific,

but, you know what, if you don't know who you want to say Hello to,

you want to say Hello to the whole world,

you can give parameters default values.

We've seen that.

Recall that with print there was a default value for SEP,

for the separator.

There was a default value for END, the line ending.

We can do that too, and here's the syntax.

If you want the value of this parameter by default,

if not provided by the programmer, to be equal to "world,"

you literally do that in the same line you're defining the function.

And I'll admit, it's starting to look more cryptic,

but I'm still just defining a function called Hello, it

takes a parameter called To, but I'm assigning it

with the equal sign a default value of "world," just in case the programmer

doesn't call Hello with an argument.

And we can see this here.

Let me change my code to use Hello in two ways.

On line 5, I'm going to very simply call Hello, no arguments.

Then on line 6, I'm going to get the name.

Line 7, I'm going to call Hello with an argument.

So you'll see Hello now being used in two ways.

Let me go ahead and run Python of Hello.py.

I'll type in my name.

Oh, interesting.

Notice I already see Hello, world, but that's

expected because line 5 happens before line 6, but once I type my name,

now the program is going to be a little more polite and say hello to me

personally.

So there too, we see with relatively simple but new syntax

how you can implement functionality very similar in spirit to what the print

function gave us automatically.

Now you have control over doing that yourself.

But let me now make this point too.

One of the whole points of defining your own functions

is one, just to avoid having to repeat yourself again and again.

You don't have to actually keep reinventing the wheel

and keep using the print function again, and again, and again,

if you just want to say Hello.

Wouldn't it be nice now if I could move this code

that I wrote for defining the Hello function, and just to be dramatic,

I'm going to hit Enter a whole lot of times, 50 lines down,

and put my definition of Hello way further down in this file.

Why?

Well, just for in the spirit of out of sight, out of mind,

because if I now rewind to the start of my program, now

you can take for granted that, oh, Hello is a function.

Why?

Because it's there on line 1 and it has an open parenthesis and a closed

parenthesis, which, up until now has meant, call this function.

And then on line 2 we're getting a variable from the user

by typing in their name and then we're calling Hello, passing in that value.

Well at this point, I can just take for granted that Hello exists,

even if it's way down further in the file or, as we'll see in future weeks,

even if it's in a different file altogether.

But there's a problem here.

And let me go ahead and run this version of Hello.py.

Notice that as soon as I run the interpreter, Python of Hello.py,

I see a name error, name Hello is not defined.

Again, did you mean Help?

Well, again, fitting.

I do need some help here, but I didn't mean to call the function Help.

The problem here, though, is that Python is just taking me literally.

I have defined my function Hello all the way down here,

but I'm trying to use it way up here.

And that's not allowed.

Python's interpreter is going to take you literally

and if you use a function, it must already

exist by the time you are calling it.

So how do I fix this?

Well, apparently I can't do that.

I have to define any functions I want at the very top of my file,

but that too could get me into a bit of trouble

eventually because if I constantly have to define a function above where I want

to use it, you're kind of writing code in reverse,

you're constantly writing functions up here, up here,

up here, as opposed to writing your code logically, top to bottom.

So let me fix this in a more standard way, which is to do this.

Generally speaking, you do want to put the main part of your code

at the top of your file.

And in fact, I'm going to go so far as to define my function, called Main.

It's not a requirement, but it's in data convention,

and this just connotes to the reader that this

is the main part of my program.

I'm going to get rid of my empty Hello call now and only pass in one version

with Hello, name.

And then down here, a couple of lines further down,

I'll actually define my Hello function.

Unfortunately, now that I've reordered the functions in this way,

by putting the main part of my code at the top and Hello at the bottom

so that my logic kind of flows top to bottom, if I go ahead and run

Python of Hello.py, Enter.

Nothing whatsoever happens.

If I do it again, nothing whatsoever happens.

Well, why in the world is this?

Well, just because I've defined a function called

Main and I've defined a function called Hello,

doesn't mean that I've actually called--

that is used-- either of them.

Yes, I'm using Hello inside of Main, but no one

is telling Python to actually use or call Main.

So in order to tidy this up, the last thing I need to do in this file,

it seems, is actually call my main function.

And, in fact, by calling my main function in this way,

it gets me out of trouble because now I'm defining Main first

but I'm not calling Hello yet.

I'm defining Hello next, but I'm not calling Hello next.

I only at the very end of this file call Main

which has the effect of running this code up here,

which has the effect of running this code down here,

and it allows me therefore to organize my file

and order my functions in any way I want, including Main at the very top,

and solving ultimately that problem of Python not knowing what's going on.

Now it's important to note that I defined my function

Hello as taking an argument To and then I

passed into that function the value of the variable

that I wanted to say Hello to, that is the variable called Name.

Because suppose I had done something a little bit differently.

Suppose that I hadn't defined Hello is taking an argument,

so I just remove mention of To and its default value "world."

And I go back up to my main function and I just

call Hello itself without passing in any arguments.

And now let me go ahead and make my one more change,

one more mistake technically, let me go ahead

and just try to naively print out the value of Name in the Hello function.

So now to be clear, in my main function on line 2,

I'm defining my variable called Name and assigning it the return value

of the input function from the user.

I'm then just calling Hello.

In my Hello function, which now no longer takes any arguments,

I am calling print, passing in Hello comma,

and then immediately passing in Name, the variable

into which I got the user's input.

But the catch is that name exists now only in Main.

And so watch what happens when I try to run this version of the program

with Python Hello.py.

I hit Enter.

I'm prompted for my name, D-A-V-I-D, Enter.

And, argh!

A name error.

Name "Name" is not defined.

So it turns out that this is actually an issue of what's called scope.

Scope refers to a variable only existing in the context in which you defined it.

So insofar as I define this variable, Name in my main function,

I can only use that variable in my name function.

I can't use it as I've tried to here in my Hello function.

It doesn't exist in that so-called scope.

And so this is why now, if I rewind and undo all of those changes,

you'll see that I'm deliberately passing Main from my main function

into my Hello function.

And now in the Hello function, it technically has a different name.

It's called To in that context, but that's fine.

It's completely up to each individual function to name its own variables

or name its own arguments, but this is a way now

that I'm handing to the Hello function the value of that variable

so it can be printed by Hello as well.

And there's one final flourish we can add here.

Now that we've implemented Hello, you'll notice that Hello only

has a so-called side effect.

It only prints out something to the screen.

Well, what if I also want my function to not have a side effect, per se,

but actually hand me back a value?

Recall that the input function returns a value,

the string that the user typed in.

Recall that the INT function returns a value.

The float function returns a value that was passed into it.

Well you can use one final keyword here, literally

Return to return a value explicitly yourself.

In fact, let me go back to VS Code here, and I

think we'll return our attention to Calculator.py

and see if we can't implement one other version of Calculator.py that actually

has our own function that even returns a value.

So I'm going to go ahead and open up calculator/py

and I think this time I'm going to throw everything away as before,

and I'm just going to start practicing what we're preaching here.

Define a function called Main which is now going

to be the main part of my function.

Let's go ahead and now declare a variable called x,

and assign it to the converted version of the user's input after asking them,

what's x?

So again, a line of code quite like we've done before.

And suppose now that what I want to do is square this value.

I want to take the number that the user has typed in

and raise it to the power of 2.

So 2 squared would be 4.

3 squared would be 9.

4 squared would be 16, and so forth.

Well how do I go about implementing a function literally

called Square which actually doesn't come with Python built-in?

Well, let me assume for the moment that it does exist, and let

me say something like this--

let me go ahead and say that printing, how

about x squared is comma square of x.

So what have I done?

I've defined a function called Main and I've implemented two lines.

The first of these lines prompts the user for a value

x and converts it to an INT and stores it in a variable called x.

On line 3, I then say x squared is and then

I pass a second argument to the print function, whatever the return

value is of a square function.

But squared doesn't exist and I'll show you this here,

if I now call Main at the bottom and I run Python of Calculator.py,

I'll see that x is 2 and then I see a whole bunch of errors, a name error,

name Square is not defined.

So this isn't a typo here, it's just the function doesn't exist.

But I think I can make it exist here.

Let me go ahead and define another function called Square.

This one's going to take in a number and I'm

going to call it generically n, as many a programmer would,

just to represent any old number.

And then what do I want to do in order to square n?

Well a number squared is really just itself times itself,

so I'm going to do this--

n times n.

But it's not enough just to do the math yourself, n times n.

You're going to have to return the actual value n times n

and that's our new keyword here.

When I now do this, watch what happens.

Python of Calculator.py, Enter.

X say shall be 2, x squared is 4.

Let me go ahead now and say x is now 3.

X squared is now 9.

So I've implemented my very own function that returns the square of a value

and because I'm using the return keyword, that

ensures that I can pass the return value of this, just

like the return value of input or INT or float, to another function,

like print instead.

And here too there's going to be so many ways to solve this same problem.

I can actually raise n to the power of 2.

We've not seen this syntax before, but if you

use two asterisks like this, two stars, that raises the thing on the left

to the power on the right.

Or it turns out there is in Python a function called

POW for raising something to the power that

takes two arguments, the first of which is the number, the second of which

is the exponent.

So there too, there's just so many ways to actually solve that same problem

as well.

So ultimately, what have we done here?

We first introduced functions-- these actions or verbs, many of which

come built into Python that you can just use in your own code.

We then introduced variables via which you could store those return values

and then maybe do something more with it.

At the end of the day too.

You now have the ability to create, to invent your own functions

to solve simple problems like Hello, or in the weeks

to come-- much more sophisticated, more challenging, more fun problems as well.

---

DAVID MALAN: All right.

This is CS50's Introduction to Programming with Python.

My name is David Malan and this is an introduction

to Visual Studio Code for CS50.

Visual Studio Code, or VS Code, is a very popular program

nowadays for writing code.

It's technically a text editor, or nowadays really

an IDE, an Integrated Development Environment,

which means it's a program that's designed

for enabling you efficiently and effectively to write code

in your computer.

Now it turns out you typically install VS Code on your own Mac or PC.

It's free, open source software, which means anyone

can download it on their own computer.

But typically, when installing it on your own computer,

particularly to use for programming with a language like Python,

there's a few other steps as well.

You need to install support for Python itself,

you might need to configure some of the lower level implementation

details of your computer just so that things work.

And so instead, for this course, we recommend that you use VS Code, yes,

but you use it in the cloud by way of CS50's own implementation

thereof on top of something called GitHub Code Spaces, which

is a hosted version of the same.

To use Visual Studio Code for CS50, go to code.cs50.io

where you'll be prompted to log in with your free github.com account.

After some number of seconds or minutes, you'll

then be redirected to your so-called code space, which

is your own virtual space in the cloud.

Somewhere out there on GitHub servers, you

will have access to your own storage space,

really your own hard drive in the cloud in which you

can create files, run programs, and even organize everything in folders.

You'll also have the ability to write code via that interface and much more.

The interface you'll see typically looks a little something

like this, whereby you'll have at the top left hand of your screen

a File Explorer that allows you to see all of the files you've

created over time in your account.

You'll then have the ability at the top to actually write code

by creating one or more tabs in which to write your programs.

And then perhaps less familiarly, at the bottom of the screen,

you're going to have access to what's called a terminal window.

A terminal window is a command line interface,

or CLI, which is very much in contrast with what

you and I are mostly familiar with nowadays,

which is a graphical user interface, or GUI.

Indeed, everything we're seeing here is graphical by nature.

There's buttons, and icons, and windows, and more.

But underneath the hood, you actually have access

not only to your own virtual hard drive in the cloud, but actually

your own server too.

Somewhere out there on GitHub servers is your own server

in a sense, your own virtual server, or technically called

a container, in which all of your own files will live.

You then have the ability to create files, run programs, even install

additional software within that server, or that container,

because that server container is itself running its own operating system.

And the operating system it's running is not

Mac OS or Windows with which you're probably familiar,

but another popular operating system called Linux.

And Linux is used by so many programmers and more nowadays

not only to write code, but to run it, on their own servers as well.

So how can you go about leveraging this new command line interface?

Well, it's worth learning just a few commands

that you might not have typed previously.

These commands work not only in Linux, but also on Mac OS,

and also with some modification potentially on Windows as well.

But if you're using Visual Studio Code for CS50 in the cloud at code.cs50.io,

you'll have the ability to run all these commands and more.

And here is just a few of those that you can type.

I'm going to go ahead now and type code, which actually is a command that comes

with VS Code itself, that enables me to create new files and open new tabs.

I'm going to change screens over to my own computer

here in which I have full screen VS Code for CS50.

So this is my browser window full screened

connected to my own code space, my virtual container, in the cloud.

And suppose that I want to go ahead and create my own first program

called hello.py, a program written in Python.

In my terminal window, I could type code hello.py and hit Enter.

But before I do, notice at top left in my so-called File Explorer,

there's technically nothing there.

There's some mention of code spaces and some mention of my own unique ID.

But there's no actual files or folders there yet.

But when I hit Enter, thereby creating a new file called hello.py,

I'm not only going to see it appear in just a moment in a tab

where I can edit it, I'm also going to see a graphical representation of it

up here, which is only to say that this environment, this integrated

environment, all works cohesively.

Here we go.

Hitting Enter.

My tab has opened, my prompt is blinking here, so I can start writing code.

But you'll also see that hello.py not only exists, but is

highlighted as the file I've opened.

Well, let me go ahead and write a very simple program now.

Print quote unquote in parentheses, hello comma world,

which as you'll see if not already is perhaps

the most formative, most canonical program

you can write first when writing code in most any language.

I'm going to go ahead now, and in my terminal window,

I'm going to run one other command that you'll

learn all the more about soon too.

I'm going to run Python itself, which is not only a language,

it's actually a program that understands that language.

And if I specify not just Python but a space, and the name of a file

that I have created containing code written in that language,

crossing my fingers as always and hit Enter, I've just run my own program.

Now on our Macs and PCs, we're generally in the habit of pointing and double

clicking on icons or on our phones, tapping on icons to launch some file

or program.

In a command line interface, or CLI, everything is done with the keyboard.

You can use your mouse to move your cursor up and down as needed.

But generally, you do everything with your keyboard,

not really with your mouse.

Everything else on the screen can be clicked on.

But in your terminal window, you'll generally

see this prompt, a single dollar sign that does not mean

dollar or currency for that matter.

It's just a standard symbol that means type your commands here.

And it's a visual indicator that you can type your next command.

And indeed, the two commands I've thus typed

are code to create my new file within VS Code itself,

and then Python, which we, CS50, already pre-installed into your code space

so that you have built in support from day one with writing your own code.

Now there's other things I can do in this environment.

I, for instance, could control click, or right click up here,

and see a whole menu of options with which to edit, or modify,

that file, just on Mac OS or Windows.

I could click on some of these icons at top left, the plus icon

to create a new file or a new folder.

So all of those GUI conventions, Graphical User Interface conventions,

are still there.

But you will and should get more comfortable over time with the command

line interface alone, which does all of those things and more for you.

In fact, here are just a few more of the commands that you can use within Linux,

and this command line interface specifically.

We'll see in a moment LS, which allows you to list all of the files

in your current folder, CP, which will allow you to copy a file from one place

to another, MV, which will allow you to move a file

or really rename it from one name to another, RM for remove,

which will allow you to delete a file, dir, sir, MKDIR, which will allow you

to make a directory, that is a folder, CD, which will allow you to change

directories from one folder to another, rmdir,

which will allow you to remove a directory, and then clear,

which will just aesthetically clear your terminal window, if like me,

you like to keep things a little bit tidy.

Now these are just some of the commands you can

type within Linux at this command line.

But they're really equivalent to all the kinds of things you and I already

do frequently on our own Macs or PCs.

Let me go back to VS Code here.

And let's just try out a few of these here.

I'm going to go ahead and first clear my screen

as though we're starting from scratch.

But this doesn't change or delete any of my files.

It just clears my terminal window.

I can alternatively hit Control L to do the exact same.

Now I'm going to go ahead and type LS.

But before I hit Enter, note that LS kind of sounds like list.

And indeed, this command and almost all of those others were fairly succinct

in that LS means list, but you only type the L and the S, no L-I-S-T.

And this is true of a lot of command line interface commands,

just to enable you and I to type things more quickly,

a lot of these commands have this very succinct syntax in names.

So here we go.

I've typed out LS already.

I'm going to go ahead and hit Enter.

And you'll see only one thing, hello.py, which

is to say when I try to list all of the files in my current default folder

that just automatically opens when I launch VS Code in the cloud,

I only see the one file exists, which I can also see graphically here.

Well, let me go ahead now and do something else.

Let's make a copy of this file and call it,

goodbye.py just so in anticipation of writing a second program that says

goodbye, world instead of hello.

I'm going to do CP for copy, then the first name of the file, hello.py, then

a space, and then the new name for the new file, goodbye.py, Enter.

And now notice nothing really has happened.

And that's generally a good thing.

If you don't see an error message down here, you've done well.

But you do see graphically up here that both files now exist.

Now I could open goodbye.py in a couple of ways.

I could actually just go up here and double click on goodbye.py

to open it here.

Or I can type the same command as earlier, code of goodbye.py,

and now I'll have a second tab open here.

And for parallelism, I could change this word

from hello to goodbye just so I have a second program

now that it is indeed the second file as well.

Now what if I decide that I don't want this file?

And I'm going to go ahead and close it here and I want to rename it

and change it to--

from goodbye.py to farewell.py.

Well, it still says goodbye.py up here.

And if I type LS down here, I'll see a list of two files now

that are in that same folder.

Let me do MV for move, goodbye.py, and then something

like farewell.py instead.

Enter.

And you'll see that it changed up here at left.

And if I type LS again, you'll see that it changed down here as well.

Well, you know what?

Let's never mind with farewell.py.

Let's go ahead and just remove this I'll altogether.

I could go up here and single click on farewell.py

in my File Explorer, control click, or right click, and then select Delete.

Or down here on my command line interface, I could do this.

I could do RM space farewell.

And now notice if I'm a little tired of typing out these long file names,

Linux and command line interfaces frequently support autocomplete.

So instead of typing farewell, I'm going to type FAR,

which sounds like more than enough characters

to finish the rest of my thought.

And I'm going to hit Tab on my keyboard.

And it just finishes it for me without having to manually type the same.

I'm going to go ahead and hit Enter.

And I'll see a somewhat cryptic message now, as many of them admittedly are.

It's not an error.

It's just a prompt.

Remove regular file quote unquote, "farewell.py".

This is just a safety check to make sure that I'm OK with deleting this file.

Generally when prompted with a question like this,

go ahead and type in Y for yes, or N for no.

And hit Enter.

And now nothing bad seems to have happened in terms of any messages.

But indeed, farewell.py is now gone.

Well, what more can we now do?

Let me clear up the screen just for aesthetics sake here.

And let's create one or more folders, not just files.

Let me go ahead and make a directory here with MKdir.

And let me just call it folder, though I could call it anything I want.

Alternatively, I could go up here to my File Explorer,

click on the plus icon here on the folder,

and manually create a no folder as well.

But here again, the goal is to focus on this command line interface instead.

All right.

MKdir folder and hit Enter.

Nothing seems to happen in the way of error messages.

But now, notice there's not just hello.py,

but a folder, or directory, called folder.

Well, what now can I do with that?

Well, let me go ahead and see what's inside.

Well, in my GUI, my Graphical User Interface,

I could click on the folder here in this little triangle, and look inside.

And of course, there's nothing there because I literally just created it.

But let me do this now in my terminal window.

Let me type LS after clicking down here.

And we'll see both my folder, which by convention has a slash at the end

to make visually clear that it's a folder, not a file,

and there's hello.py.

Let me go ahead now and change directory.

Now in Mac OS or Windows, we would again just click or double click

on the folder and see what's inside.

In a command line interface, you need to be more deliberate.

CD space folder, with or without the slash is fine, and then hit Enter.

And now you'll see that your prompt changed.

There's still a dollar sign.

But by convention, and this is CS50 detail,

though other people do this too in the world,

we have prefixed your prompt with a just quick reminder

of what folder you're in so that you don't forget whether you're

in your main folder where you just began, or some other folder therein.

I'm going to go ahead now and type LS to list the contents of this folder.

And of course, there's nothing there because I haven't created any files.

But if I do this, I could do code of farewell.py again to bring it back.

But this time in this folder, hitting Enter,

you'll see a new tab opens here where I can write some more code.

But you'll also see hierarchically here, the farewell.py

is now inside of this folder.

Now suppose this is a mistake and I didn't mean to do that.

What can I do?

Well, let me close the tab here, even without writing any code.

And down in my terminal window, let me do this.

I'm going to type LS to confirm that, yep, farewell.py exists there.

But suppose I want to move it into my main folder.

Well, like on Mac OS or Windows, I could click on the file here, farewell.py,

and just kind of drag it up or down to put it where I want.

But let's move it, not rename it this time,

but move it using that same MV command instead.

I want to move farewell.py to the parent folder.

In the world of files and folders, you can think of things like a family tree.

And if you want to go up hierarchically, you want to go into the parent folder,

so to speak, whereas folder is conversely a child of the default

folder in which we began.

So I want to move farewell higher up.

How do I express that?

Well, another piece of syntax in a command line interface

is typically this, dot, dot.

Dot, dot is a symbolic representation of whatever the parent directory

is, no matter what its name is.

In this case, it's my own default folder.

Doesn't even matter what its name technically is.

It's not pictured on the screen.

I'm going to go ahead and hit Enter.

And watch what happens at top left in the graphical version of the same.

We should see that farewell.py now pops out of the folder

and the indentation changed to imply that it's not inside of the folder now.

It's alongside hello.py.

Indeed, if I close the folder here, I don't see a difference

because there's nothing in there as well.

Let me type LS to confirm in my terminal window.

There's nothing in folder.

Let me now go back to my parent directory.

And I can do this in different ways.

And like with programming itself, there's

going to be many ways sometimes that you can navigate a command line interface.

Here's one.

I can do CD for change directory, and then space dot,

dot to go back to the parent directory.

But another technique, especially if you're

a little less comfortable with this whole new world,

is whenever in doubt, if you want to go back

to your default directory in which you began when you logged in,

just type CD with nothing after it and hit Enter.

And you'll be whisked away back to the starting point.

Indeed, if I type LS now, I'll see farewell.py, my folder if empty,

and hello.py as well.

Now, I don't really need this folder because there's nothing in it anymore.

So let's go ahead and delete that.

I could click on it with control click or right click in the File Explorer.

And I could delete it manually like in Mac OS or Windows.

Or I can use my newfound commands, rmdir for remove directory, space folder,

with or without the slash, and hit Enter.

And because it's empty, I'm not even prompted.

It just goes away quite cleanly.

And a final technique that you'll find quite helpful

in the context of a command line interface

is that it tends to store your whole history of commands.

In fact, not just autocomplete.

But if I hit up, and up, and up, and up, you'll

see my history of all previous commands scrolling up on there

on the screen, which is helpful if I just want to execute one of them again.

I don't need to manually type it out.

I can scroll up and down and find the one that I want.

So this then is just the scratching of the surface

of a command line interface in the context of Visual Studio

Code in the cloud.

Most of these commands would be the same whether on your Mac or PC, at least

if you're using a terminal window that's connected to a computer running Linux,

be it a container, or a virtual machine, or an actual server somewhere

in the cloud.

In the weeks to come for this class, will you have access to code.cs.cs50.io

and in turn your own code space so that you have access 24/7 365 via a browser

alone so long as you have internet access

without needing to install anything on your own Mac or PC.

Toward the end of the semester, if you're feeling more comfortable

and would like to use this same software,

VS Code, if not Linux as well, you'll be able to also install

those for free on your own Mac or PC and continue programming

onward even after the courses own end.

This then was Visual Studio Code for CS50,

and with it, a command line interface.

And this is CS50 itself.