Getting Started with Python: Introduction

This 15-video course lets learners explore the basics of how to use the Python programming language. You will learn to set up with an interactive environment that allows you to develop and run Python scripts on your machine. Begin by installing Anaconda, an open-source distribution of the Python and R programming languages. You will learn to write your first meaningful program in Python, then create a Jupyter notebook, the most popular tool for writing and running Python code. You will learn how to do simple coding by using Python's Jupyter notebooks, and explore different Jupyter functionalities, including built-in functions. Learners will explore how to use a Python variable to store values, and learn to differentiate between variables of different types, and the different ways to assign values to variables. You will examine how variables act as containers, and you will learn how to change values that are inside a container. Finally, you will learn to use integers, floating-point numbers, strings, and to work with Boolean values.

Table of Contents

Course Overview Python Introduction

[Video description begins] *Topic title: Course Overview* [Video description ends]

Hello there, welcome to this course, Getting Started with Python Programming. My name is Janani Ravi and I will be your instructor for this course.

[Video description begins] *Your host for this session is Janani Ravi. She is a Software engineer and big data expert.* [Video description ends]

A little about myself first. I am a co-founder at Loonycorn, a studio for high-quality video content. Before founding Loonycorn, I worked at various companies including Google for seven years in New York and Singapore and Microsoft. And before that, I attended grad school at Stanford. Python is fast becoming more than a programming language. It is now the gateway into the world of big data, machine learning, and artificial intelligence.

Virtually all of the most important deep learning frameworks are Python-based and Python is also the language of choice for those who are new to programming. In this course, you will get started with Python by installing and getting started with the software, as well as writing your first meaningful program and exploring important language constructs.

There are no prerequisites for this course. This is one that can be used by anyone looking to begin their fascinating journey as programmers. In this course, you will begin by installing the Anaconda distribution for Python and getting started with simple coding using Jupyter Notebooks. Jupyter Notebooks are the most popular tool for writing and running Python code these days. And we will install Jupyter and explore some of its functionality.

Then, you will move on to exploring some of the crucial concepts that are essential for you to accomplish meaningful tasks in Python. Working with common built-in functions, defining variables, differentiating between variables of different types, and working with strings.

By the end of this course, you will be set up with an interactive environment that allows you to develop and run Python scripts on your machine. Such an interactive environment is a powerful development aid. And you will use it to use built-in functions, define variables, and create and manipulate simple as well as multiline strings.

Install and Set up Anaconda on Windows for Python

[Video description begins] *Topic title: Install and Set up Anaconda on Windows. Your host for this session is Janani Ravi.* [Video description ends]

So, you've just started learning to code and you've chosen Python as your programming language. That's a great choice. But how do you get started coding in Python? Well, you need to download and install the Python programming language on your machine so that you can code with it.

And that's exactly what we'll do using the Anaconda distribution. The install steps followed in this video are if you have a Windows machine. Simply type in onto your browser window, install Anaconda on Windows. And this will bring up a number of hits on your favorite search engine.

[Video description begins] *A browser window appears. She opens Google.com and types install anaconda on windows in the search field. The search results display below.* [Video description ends]

What is Anaconda and what are all these serpent references? You know that Python is a programming language. Anaconda is an open-source distribution of the Python and R programming languages. Open source means that they're completely free for any developer to download and use.

Here, we're not really interested in R, we'll focus our attention on Python. When we use the Anaconda distribution, not only do we get the Python programming language, we also get a package called Conda. Conda is what is termed as a package manager. Now, you might be aware that when you're working with Python, you have dozens of libraries that you can use to solve common problems.

Now, these range from very complicated libraries for machine learning and data science to simple libraries for array manipulation to libraries that allow you to download content from websites, scrape the web, etc. The package manager is what allows you to install these libraries easily. It does all of the difficult heavy lifting for you behind the scenes.

With this brief introduction of the Anaconda distribution and the Conda package manager out of the way, let's get set up on Windows with Python using the Anaconda distribution. Click on this link here which shows you how you can get Anaconda on Windows.

[Video description begins] *From the search results, she clicks the link saying Installing on Windows- Anaconda 2.0 documentation. Its URL is https://docs.anaconda.com/anaconda/windows/.* [Video description ends]

Here we are on the install page. Observe that the Anaconda distribution can be installed on any operating system, Windows, Mac OS, Linux, and so on.

[Video description begins] *The Anaconda documentation website displays. In the left pane display various options such as Home and Anaconda Distribution. Anaconda Distribution is selected by default. The right pane shows a page titled Installing on Windows. The page contains the steps to be followed to install Anaconda on a Windows machine*. [Video description ends]

If you have a different operating system, you can simply select the link for your OS. In this particular course, we will cover installing Python with the Anaconda distribution on Windows as well as on the Mac.

[Video description begins] *Under the Anaconda Distribution option in the left pane, various options such as Installing on Windows, Installing on Linux, and Installing on Linux POWER display. She highlights the options, Installing on Windows and Installing on macOS*. [Video description ends]

Click on the link which says Download the Anaconda installer. This will take you to the install page which should automatically detect your operating system.

[Video description begins] *From the right pane, she clicks the hyperlink, Download the Anaconda installer, given in the first step*. [Video description ends]

But sometimes that doesn't really work. So, I'm going to go ahead and select my operating system here.

[Video description begins] *She scrolls down the page which displays and reaches the Anaconda 2018.12 for macOS Installer section. Just above the section, the logos for Windows, macOS, and Linux display. She highlights the three operating systems and a blue arrow points to Windows*. [Video description ends]

I'm going to select on the Windows tab here which will give me the right software for my Windows machine. Select the Windows option and here we have it. This is the latest version of the Anaconda software for Windows available at the time of this recording.

[Video description begins] *As she clicks the Windows option, a section titled Anaconda 2018.12 for Windows Installer displays below. The title is highlighted and the section contains two sub-sections named Python 3.7 version and Python 2.7 version. A Download button displays in both sub-sections along with their hardware requirements*. [Video description ends]

Now, the first thing that will jump out at you when you look at this page is the fact that there are two versions of Python, Python 3.7 and Python 2.7. Why are there two versions and which one should you chose? Now, this is something that newbie programmers often get tripped up with. Now, Python has been around a long time and there are many older projects build using the older Python 2.7 version.

It's still fully supported and widely used. There is no difference in the fundamental programming concepts and the language itself between the two versions. But there are many subtle differences that can trip up a new programmer. If you're starting afresh and you have no old code to deal with, you should start with Python 3. That is the Python version that will be supported in the future and all new projects use Python 3.

[Video description begins] *She highlights Python 3.7 version*. [Video description ends]

All of the programs that we'll write in this course and on other courses in this path will use Python 3. With this in mind, let's go ahead and select the Download button here for Python 3.7. This is the latest version of Python at the time of this recording. Selecting Download will bring up a download window and you can select where you want this EXE file to be located.

[Video description begins] *A Save As dialog box displays. The File name field contains Anaconda 3-2018.12-Windows-x86_64 and the Save as type field contains Application. The Documents location is open now*. [Video description ends]

Choose the download location that makes sense for you and click on the Save button to download this application. Once the executable to install the Anaconda software on your machine has been downloaded, you

can go ahead and open it up and start the installation process.

[Video description begins] *She right-clicks the downloaded file in the task bar and a shortcut menu appears containing options such as Open and Show in folder.* [Video description ends]

And this will bring up this really simple dialog where you simply follow instructions on this dialog in order to get Anaconda and Python set up on your machine.

[Video description begins] *The Anaconda setup wizard displays in a window. The first step page is titled Welcome to Anaconda3 2018.12 (64-bit) Setup. The Next and Cancel buttons display at the bottom of the window.* [Video description ends]

Click on Next through every screen. Make sure you agree with the license agreement. Click on the I Agree button here and move on to the next page.

[Video description begins] *The License Agreement page displays in the window and three buttons, Back, I Agree, and Cancel display at the bottom. She clicks I Agree.* [Video description ends]

Now, your machine might have multiple users. Here is where you make the choice of whether you want this Anaconda distribution and Python installed for all users or only for yourself.

[Video description begins] *The Select Installation Type step displays. It contains two radio options labeled Just Me and All Users. The Just Me option has recommended written against it in parenthesis and the All Users option has requires admin privileges written against it.* [Video description ends]

If you install for all users, you need to be an administrator on your machine. Since I am the administrator here, I'm going to go ahead and select All Users. Click on the Next button to proceed with the installation. Decide where the destination folder for this installation should be. If that's okay with you, click on Next once again and move on.

[Video description begins] *The Choose Install Location step displays. The Destination Folder field contains C:\ProgramData\Anaconda3 and a Browse button next to it. The Space required for installing Anaconda is mentioned as 2.8GB. She clicks the Next button.* [Video description ends]

This will take you to a page which shows you some advanced installation options. Now, the particular detail that you need to pay attention to is whether you want to use this distribution of Python as the Python version used by the system as a whole.

What does this mean? Now, you might have other versions of Python installed on your local machine. If you click and select this check box, this version of Python that we are installing using Anaconda will be the version that applies by default.

[Video description begins] *The Advanced Installation Options step displays. It contains two check boxes labeled Add Anaconda to the system PATH environment variable and Register Anaconda as the system Python 3.7. She checks the second check box.* [Video description ends]

Well, that's exactly what I want. I want to make sure that we do all of our demos using this latest version of Python. So, I am going to select this check box and click on the Install button.

[Video description begins] *The Installing page displays. It contains a progress bar and a Show details button. All the buttons on this page are disabled now as the installation is not complete.* [Video description ends]

And now, just wait for a few seconds till the installer does its job and completes the installation of the Anaconda distribution on your machine. Click on the Next button. This will show you the options for additional software packages that you can install on Windows.

[Video description begins] *When the progress bar becomes green, the Next button gets enabled and the top of the window says Installation Complete*. [Video description ends]

[Video description begins] *The next page contains an Install Microsoft VSCode button and a hyperlink, Visual Studio Code License. She clicks the Skip button*. [Video description ends]

Let's skip this. We're only focused on Anaconda and Python. And let's move on to the next page. And here we are at the end of our installation. You now have Python on your local machine. It's as simple as that.

[Video description begins] *The window now displays the text, Thanks for installing Anaconda3!. The page contains two check boxes labeled Learn more about Anaconda Cloud and Learn how to get started with Anaconda. Both check boxes are checked. She clicks the Finish button*. [Video description ends]

Click on the Finish button here. And you've successfully installed and set up Python using the Anaconda distribution.

Run Jupyter notebooks on Windows for Python

[Video description begins] *Topic title: Run Jupyter notebooks on Windows. Your host for this session is Janani Ravi*. [Video description ends]

And let's move on and see how we can interact with Python. And for this, you need to bring up the command prompt on your Windows machine. Use the Start menu or the search option to bring up the command prompt. I'm going to search for c. Here is the command prompt.

[Video description begins] *She clicks the Start menu and searches for command prompt. A list of suggestions starting with the letter c display. She clicks Command Prompt under the Apps group*. [Video description ends]

Select the command prompt and it'll open up a terminal shell window.

[Video description begins] *The shell window opens. The command prompt reads C:\Users\admin*. [Video description ends]

I'm going to go ahead and switch to full screen for this particular command prompt so that it's easier for us to view the commands that we type in. Here we are in the full screen of our command prompt.

[Video description begins] *She clicks the Maximize button in the top-right corner of the window and the window expands to full screen*. [Video description ends]

Now, I'm going to create a directory where I'm going to store all of my demo files that I create in Python. Let's create a new folder here, you can call it anything. I'm simply going to call it demos.

[Video description begins] *At the command prompt, she types the following command: mkdir Demos*. [Video description ends]

MD Demos and CD into the demos directory.

[Video description begins] *She types the following command: cd Demos*. [Video description ends]

You can confirm whether Python has been installed successfully on your machine by simply running the command python.

[Video description begins] *The command prompt changes to C:\Users\admin\Demos. She types the following command: python*. [Video description ends]

Notice that the prompt has changed. You are now within the Python interpreter.

[Video description begins] *The output displays the current version of Python, Python 3.7.1*. [Video description ends]

Now that we know that Python has been installed successfully, type in exit and get out of the Python shell.

[Video description begins] *She types the following command: exit ( )*. [Video description ends]

This is not where we will be writing our Python code. We will use something simpler, more intuitive, and more fun. In order to work with Python, you need to be within the Conda Package Manager environment. And for this, you can activate the default environment on your Windows command prompt. Simply type in activate base. Base is the name of the default virtual environment.

[Video description begins] *She types the following command: activate base*. [Video description ends]

You don't really need to worry about the nitty-gritty details of what a virtual environment is, why it's needed, and so on. Remember that this will allow you to work with the Conda Package Manager, work with Python, and bring up Jupyter Notebooks.

[Video description begins] *She types the following command: jupyter notebook*. [Video description ends]

So, what exactly is a Jupyter Notebook? Well, this is the interactive shell that we'll use to type out our Python code. Now, the reason why Python is so popular with students who are getting started with programming is because of its ease of use.

And an important factor that contributes to its ease of use is this amazing IDE or Integrated Development Environment provided by Jupyter Notebooks. Jupyter Notebooks offer a browser-based interactive shell where you can type in your Python commands within cells.

Execute those cells and see the results of your execution right there within your browser. If you type in the jupyter notebook command, this will bring up a URL that should automatically open up within your browser tab. If it doesn't, you simply copy-paste this URL onto a new browser window.

[Video description begins] *The URL is file://C:/Users/admin/AppData/Roaming/jupyter/runtime/nbserver-14480-open.html*. [Video description ends]

You will have to leave this command prompt open and have this program running. This is the Jupyter Notebook server. So, don't kill or close this command prompt because you'll need this to work with Jupyter Notebooks.

When you're done programming, you can kill this command and close this command prompt. And this should bring up a browser window that looks like this. This is the home page for Jupyter where we'll create what are called Jupiter Notebooks where we'll write our Python code.

[Video description begins] *The Jupyter Notebook interface opens in a browser window. The Jupyter logo displays in the top-left corner and the Quit and Logout buttons display in the top-right corner. Underneath, three tabs, Files, Running, and Clusters display. The Files tab is active now. The Upload, New, and Refresh buttons display below the tabs, in the right corner. Underneath, the Name, Last Modified, and File size buttons display. The notebook is empty now.* [Video description ends]

The files that you create using Jupyter Notebooks will be automatically saved within your current working directory or your current folder and they will have the ipynb extension. Now, these files are referred to either as Jupyter Notebooks or iPython Notebooks. iPython is an older name, which is still often used. In order to create a new notebook, you need to click on this new menu drop-down to the top right of your screen. Click on New. This will bring up a menu, and you can select the Python 3 option.

[Video description begins] *When she clicks the New button, a drop-down menu displays. It contains the Python3 option along with other options such as Text File, Folder, and Terminal. She selects Python3.* [Video description ends]

Selecting Python 3 indicates that the notebook will run on the Python 3 kernel. So, the code that you write will be interpreted and executed using the Python 3 kernel. This will open up a new notebook on a new tab on your browser and this notebook will automatically get the name Untitled.

[Video description begins] *A new Jupyter notebook titled Untitled displays in a new tab. Next to the title, the text, Last Checkpoint: a few seconds ago (unsaved changes), displays. Underneath, a Menu Bar displays containing menus such as File, Edit, and View. The right corner of the Menu Bar contains the text, Kernel starting, please wait.... Below the Menu Bar, a toolbar containing many tools such as Run, Stop, and Refresh displays. The main interface contains a long rectangular box preceded by In [ ]. The cursor is blinking in the box.* [Video description ends]

It's extremely easy to change the name and rename this notebook so that it has a name that is meaningful, based on the demo or the Python code that you're going to write. Select the current name of this notebook, that is, Untitled. Click on it, and this will bring up a little dialog, and then, you can rename this notebook to be anything.

[Video description begins] *The dialog box is titled Rename Notebook and contains a single field labeled Enter a new notebook name. The Cancel and Rename buttons display at the bottom of the box.* [Video description ends]

Let's say you want to call it MyFirstPythonNotebook. Click on the Rename button and you'll find that your notebook name has been updated. You are now ready to get started with coding. But how do you code? Well, this little box that you see here on screen is actually a code cell and you type in your code within this little box. And then, you hit Shift + Enter in order to execute this code.

[Video description begins] *In the code cell, she types the following command: !python --version.* [Video description ends]

Shift + Enter is what you hit, both of these keys together, to execute the command that you've typed into the code cell. Shift + Enter will execute those commands, print the results on screen, and it'll also create new code cells if needed. Observe that the command that we've typed in here, is not really a Python command. Instead, it's a shell command or something that you would type in on your command prompt.

You can also type in your command prompt commands within your Jupyter notebook. Make sure that you precede these commands with an exclamation point. This will tell your notebook that this is not a Python command; instead, it's a shell command. So, we have !python -- version, that will print out the current version of Python that we're using, which is Python 3.7.1.

[Video description begins] *As she presses Shift and Enter keys, the output displays in the space below the code cell and a new code cell gets added after that. The number one displays within the square brackets of the first code cell. The output of the !python --version command is highlighted.* [Video description ends]

Install and Set up Anaconda on MacOS for Python

[Video description begins] *Topic title: Install and Set up Anaconda onMacOS. Your host for this session is Janani Ravi.* [Video description ends]

So, you've just started learning to code and you've chosen Python as your programming language. That's a great choice. But how do you get started coding in Python? Well, you need to download and install the Python programming language on your machine so that you can code with it. And that's exactly what we'll do using the Anaconda distribution.

The install steps followed in this video are to set up Python on your Mac machine. So, follow these steps if you have a Mac machine. Type in install anaconda on mac and this will take you to the search results on your favorite search engine.

[Video description begins] *A browser window appears. She opens Google.com and types install anaconda on mac in the search field. The search results display below.* [Video description ends]

Now, the first thing that might have struck you here is that there are a lot of snake references. Python is a snake as is Anaconda. So, you know that Python is a programming language. What exactly is Anaconda? Anaconda is the open-source distribution of the Python as well as R programming languages.

Open source, it's absolutely free for any developer to download and use. Within the Anaconda distribution, not only do you get Python and R but bunch of other tools that makes it very easy for a new programmer to get started. Now you might be aware that the Python programming language has a number of libraries and packages to perform a bunch of common tasks. So, you don't have to write the code for everything yourself.

There are packages which allow you to perform data science, data wrangling, machine learning operations, everything. There are Python libraries that make it easy to build web apps to scrape websites, whatever you need to do.

Within the Anaconda distribution, you have something called the Conda Package Manager, which allows you to download and install Python packages and libraries that you might use. Now, let's move on and let's get Anaconda set up on our local machine. Click on this link which says Installing on macOS. Anaconda 2.0 is the version that we are going to use.

[Video description begins] *From the search results, she clicks the link saying Installing on macOS - Anaconda 2.0 documentation. Its URL is https://docs.anaconda.com/anaconda/mac-os/.* [Video description ends]

This will take you to the install page for Anaconda and on the left navigation menu, you can see that the Anaconda distribution is available for a number of different operating systems, Linux, Windows, as well as the Mac.

[Video description begins] *The Anaconda documentation website displays. In the left pane display various options such as Home and Anaconda Distribution. Anaconda Distribution is selected by default. The right pane shows a page titled Installing on macOS. The page contains the steps to be followed to install Anaconda on a Mac machine.* [Video description ends]

Click on the link here that says macOS installer and this will take you to a web page where you can download the Anaconda distribution of Python.

[Video description begins] *From the right pane, she clicks the hyperlink, macOS installer given in the first step.* [Video description ends]

This will take you to a page where the Mac OS versions for Python are available. And the first thing that you're faced with is a choice.

[Video description begins] *The page with the URL, https://docs.anaconda.com/anaconda/mac-os/ displays. The page is titled Anaconda 2018.12 for macOS Installer. The logos for Windows, macOS, and Linux display at the top of the page and she highlights macOS there .* [Video description ends]

There are two versions of Python here, a Python 2.7 version and a Python 3.7 version. So which one do you choose? Python 2.7 is the older version and often referred to as Python 2.

[Video description begins] *Underneath the title, the page contains two sub-sections named Python 3.7 version and Python 2.7 version. A Download button displays in both sub-sections along with their hardware requirements.* [Video description ends]

Python has been around a long time and there are many projects in the real world built using Python 2.7. And that version is actively supported. However, if you're starting afresh, it's best to use Python 3. The programming language itself and the concepts involved are not very different whether you choose Python 2 or 3.

But there are subtle differences in certain functions, especially the print statement that can trip up newbie programmers, so it's best to stick with Python 3. That's the Python version of the future. All of the demos in this course will be in Python 3. Go ahead and select the Python 3.7 version.

Click on this Download button here and the .pkg file will be downloaded onto your local machine. Once the download is complete, you can click on this drop-down and open up this installer to get Python installed on your Mac machine.

[Video description begins] *She right-clicks the downloaded file in the task bar and a shortcut menu appears containing options such as Open and Show in folder. A notification saying This package will run a program to determine if the software can be installed displays. The Cancel and Continue buttons display in the notification pop up*. [Video description ends]

Click on Continue here. We do want to run Anaconda which has the Python programming language that we want to work with. Now, the actual process of installing Anaconda is very straightforward. Simply click on Continue. Here are all of the packages that will be installed. You don't need to worry about these at this point in time.

[Video description begins] *The Install Anaconda3 wizard displays in a dialog box. All the steps in the set up process are listed in the left pane. The steps are Introduction, Read Me, License, Destination Select, Installation Type, Installation, Microsoft VSCode, and Summary. The right pane displays some text under the heading, Important Information. Buttons labeled Print, Save, Go Back, and Continue display at the bottom of the box*. [Video description ends]

Let's move ahead and click on Continue. Accept the license agreement. That's, of course, a necessity. If you click on the Continue button here, this will bring up a dialog where you can select Agree. You've agreed to the license agreement. You can proceed with the installation of Anaconda 3.

[Video description begins] *She keeps clicking the Continue button and the relevant step gets highlighted in the left pane. When she reaches the Installation Type step, the Continue button is replaced by an Install button*. [Video description ends]

Click on the Install button and type in your admin password. You'll need that to complete the install and keep moving forward. Click on the Install software button here.

[Video description begins] *When she clicks the Install button, a pop-up containing the Username and Password fields displays. She types Loonycorn in the Username field and a password. Then, she clicks the Install Software button in the pop-up*. [Video description ends]

And this should get you set up with Anaconda and Python.

[Video description begins] *The installation progress is shown by a progress bar. When the installation process is complete, the Microsoft VSCode step page opens*. [Video description ends]

You can go ahead and skip the step which asks you to install Microsoft VS code. Click on Continue once again. This will bring up an option that asks whether you want to move your installer to trash. You no longer need it now that you have Anaconda set up. You can choose either of these options, both are okay.

[Video description begins] *The options are Keep and Move to Trash*. [Video description ends]

I'm going to choose to keep the installer around even though I've completed installation.

Run Jupyter notebooks on MacOS for Python

[Video description begins] *Topic title: Run Jupyter notebooks on MacOS . Your host for this session is Janani Ravi*. [Video description ends]

We are now ready to get started coding in Python. But how do we do that? What do we use? Well, bring up a terminal window on your Mac machine. Here is my terminal window. My prompt might look a little different but that's because of the customization that I've used.

[Video description begins] *The command prompt terminal screen displays*. [Video description ends]

This is just a plain, old terminal window. On the terminal window, run the python --version command in order to see the current version of Python that is running on your machine that has been installed. You can see that it's 3.7.1. This is the version that we just installed using Anaconda.

[Video description begins] *She types the following command: python --version. The output displays and is highlighted*. [Video description ends]

If you run the command whereis python, you'll find that there is a version of Python available under /usr/bin/python. Now, it's important that you remember that this is not the version of Python that we're going to use. This is the Python version that comes built in on your Mac.

[Video description begins] *She types the following command: whereis python. The output is /usr/bin/python*. [Video description ends]

So, if we hadn't installed Python using Anaconda, this is the version of Python that would be running on your machine. But because we do have Python installed using Anaconda, when you run which python, you'll see the version of Python that our machine will use to execute our programs. This is the Python present under /anaconda3/bin/python.

[Video description begins] *She types the following command: which python. The output is /anaconda3/bin/python*. [Video description ends]

So, even though we have multiple Pythons installed, this is the Python that will be used, the one in anaconda3. Here I am in the home directory of my Mac machine. I'm going to run an ls command here to see what

[Video description begins] *She types the following command: ls*. [Video description ends]

directories I have under here. Now, here is the Projects directory and within the Projects directory, I've already created this sub-folder called Skillsoft. And within Skillsoft, I have the SkillsoftBasicPython sub-folder.

[Video description begins] *Many directories display in the output. They include the Projects directory. Next, she types the following command: cd Projects/Skillsoft/SkillsoftBasicPython/*. [Video description ends]

This is the sub-folder that I'll use to write code. This is, of course, up to you. You can choose any directory or folder on your local machine to get started.

[Video description begins] *The command prompt changes to Projects/Skillsoft/SkillsoftBasicPython/*. [Video description ends]

But where do we write our Python code? We'll use a tool or an IDE, an Integrated Development Environment called Jupyter Notebooks. You can learn more about Jupyter Notebooks at jupyter.org. But basically what it is is a browser-based interactive shell where you can type in code right within your browser window, execute that code, and see the results of your execution right there within the browser.

[Video description begins] *She types the following command: jupyter notebook*. [Video description ends]

Now this is extremely cool because it gives you fast feedback as to whether you've made a mistake or not. Jupyter Notebooks is one of the most popular environments that Python developers use to write and prototype their code. Running Jupyter Notebook will run the Jupyter Notebook server right here within your command prompt. So, make sure that you don't close this terminal window.

The notebook server that we just ran will automatically open up a browser window. Or if it doesn't, you can simply copy paste this URL into a new browser window and that's what I will do here. And this brings up the home page for Jupyter Notebooks and this should bring up a browser window that looks like this.

[Video description begins] *The Jupyter Notebook interface opens in a browser window. The Jupyter logo displays in the top-left corner and the Quit and Logout buttons display in the top-right corner. Underneath, three tabs, Files, Running, and Clusters display. The Files tab is active now. The Upload, New, and Refresh buttons display below the tabs, in the right corner. Underneath, the Name, Last Modified, and File size buttons display. The notebook is empty now.* [Video description ends]

This is the home page for Jupyter where we'll create what are called Jupyter Notebooks where we'll write our Python code. The files that you create using Jupyter Notebooks will be automatically saved within your current working directory or your current folder. And they will have the ipynb extension.

Now, these files are referred to either as Jupyter notebooks or IPython notebooks. IPython is an older name which is still often used. In order to create a new notebook, you need to click on this New menu drop-down to the top right of your screen. Click on New. This will bring up a menu and you can select the Python 3 option. Selecting Python 3 indicates that the notebook will run on the Python 3 kernel.

[Video description begins] *When she clicks the New button, a drop-down menu displays. It contains the Python3 option along with other options such as Text File, Folder, and Terminal. She selects Python3*. [Video description ends]

So, the code that you write will be interpreted and executed using the Python 3 kernel. This will open up a new notebook on a new tab on your browser. And this notebook will automatically get the name Untitled.

[Video description begins] *A new Jupyter notebook titled Untitled displays in a new tab. Next to the title, the text, Last Checkpoint: a few seconds ago (unsaved changes), displays. Underneath, a Menu Bar displays containing menus such as File, Edit, and View. The right corner of the Menu Bar contains the text, Kernel starting, please wait.... Below the Menu Bar, a toolbar containing many tools such as Run, Stop, and Refresh displays. The main interface contains a long rectangular box preceded by In [ ]. The cursor is blinking in the box*. [Video description ends]

It's extremely easy to change the name and rename this notebook so that it has a name that is meaningful based on the demo or the Python code that you're going to write. Select the current name of this notebook, that is Untitled, click on it, and this will bring up a little dialog. And then, you can rename this notebook to be anything.

[Video description begins] *The dialog box is titled Rename Notebook and contains a single field labeled Enter a new notebook name. The Cancel and Rename buttons display at the bottom of the box*. [Video description ends]

Let's say you want to call it MyFirstPythonNotebook. Click on the Rename button and you'll find that your notebook name has been updated. You're now ready to get started with code. But how do you code? Well, this little box that you see here on screen is actually a code cell.

And you type in your code within this little box, and then, you hit Shift+Enter in order to execute this code. Shift+Enter is what you hit, both of these keys together, to execute the command that you've typed into the code cell. Shift+Enter will execute those commands, print the results on screen, and it will also create new code cells if needed.

[Video description begins] *In the code cell, she types the following command: !python --version*. [Video description ends]

Observe that the command that we've typed in here is not really a Python command. Instead, it's a shell command or something that you would type in on your command prompt. You can also type in your command prompt commands within your Jupyter Notebook. Make sure that you proceed these commands with an exclamation point.

This will tell your notebook that this is not a Python command, instead it's a shell command. So, we have !python --version that will print out the current version of Python that we are using which is Python 3.7.1.

[Video description begins] *As she presses Shift and Enter keys, the output displays in the space below the code cell and a new code cell gets added after that. The number one displays within the square brackets of the first code cell. The output of the !python --version command is highlighted*. [Video description ends]

Using Python as a Calculator

[Video description begins] *Topic title: Using Python as a Calculator. Your host for this session is Janani Ravi*. [Video description ends]

Here we are on a brand new Jupyter notebook ready to get started with coding our first Python program.

[Video description begins] *A Jupyter notebook titled UsingPythonAsACalculator displays. The notebook title is highlighted*. [Video description ends]

The name of the notebook here is UsingPythonAsACalculator. And what you're going to see is, at its heart, a programming language is also a calculator which performs calculations or computations. It, of course, can perform far more complex computations than an ordinary calculator can.

We spoke earlier about hitting Shift+Enter to create new code cells where you can write and execute commands. You can also use this + button here in this Jupyter notebook menu in order to create code cells and that's what I'm going to do here.

[Video description begins] *She highlights the Plus button in the toolbar and hovers over it*. [Video description ends]

Click on + multiple times and a new code cell will be created each time. These are the code cells where we'll write our Python code.

[Video description begins] *She clicks the Plus button a number of times and a new code cell gets added below the previous one on each click*. [Video description ends]

Before that, let's confirm the Python version that we're using. We saw earlier that you can run shell commands or command prompt commands from within your Jupyter notebook. Make sure that you precede this command using an exclamation point. You can see the current version of Python is 3.7.1.

[Video description begins] *A cell in the form of an empty input box displays with the command prompt: In [ ] : to the left of it. The cursor blinks in this input box. Here, she types the first command: !python --version. She clicks the Shift and Enter keys and the following output displays below: Python 3.7.1. The number 1 now appears within the square brackets after the word 'In' in the command prompt next to the first input box*. [Video description ends]

The way you execute any command within a code cell is by hitting Shift+Enter. Let's type in some simple Python code. I want to calculate 7 + 9. So, I simply type in 7 + 9 and notice that when I hit Shift+Enter I get the output here right on screen within my browser window. The result is 16.

[Video description begins] *In the input box of the next code cell, she types the following command: 7 + 9. The number 2 now appears within the square brackets after the word 'In' in the command prompt next to the second input box. Then, she clicks the Shift and Enter keys and the output, 16, displays below, after Out [2]:.* [Video description ends]

Even though you aren't using any Python-specific commands, what you've actually executed is a bit of Python code. When you hit Shift+Enter, the notebook computed the result of this + operation and displayed the result within your browser window. Let's try this once again. 20 + 47. Hit Shift+Enter. The result is 67.

[Video description begins] *She highlights the plus symbol in the command, 7 + 9. In the input box of the next code cell, she types the following command: 20 + 47. The number 3 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 67, displays below, after Out [3]:.* [Video description ends]

So far we've used just integers; we haven't worked with decimals or floating point values yet. You can include floating point values in your calculations as well such as 27.5 + 9.4 + 2.

[Video description begins] *In the input box of the next code cell, she types the following command: 27.5 + 9.4 + 2. Blue colored arrows point to the three numerical values in the command.* [Video description ends]

Floating point values such as decimals as well as integer values are together referred to as numeric values. Shift+Enter will give you the result of this addition which is 38.9.

[Video description begins] *The number 4 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 38.9, displays below, after Out [4]:.* [Video description ends]

It's not just addition operations that are supported by Python. Python supports subtraction as well, as well as division, multiplication, and so on. 17-10 will give you 7.

[Video description begins] *In the input box of the next code cell, she types the following command: 17-10. The number 5 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 7, displays below, after Out [5]:.* [Video description ends]

Now that you understood how to apply arithmetic operations within Python, you can try this with a number of different examples. Here is an example of subtraction once again. This time we've used floating point numbers, and that works as expected.

[Video description begins] *In the input box of the next code cell, she types the following command: 23.6-13.6. The number 6 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 10.000000000000002, displays below, after Out [6]:.* [Video description ends]

Now you might see that 23.6 -13.6 is actually just 10 and you'll be right. With floating point numbers, the precision might be a little off. So, this number, the result here is very close to 10 not 10 exactly. This is true of all programming languages not just Python. Very high precision might be a little tricky. If you can perform simple addition and subtraction, it stands to reason that you can combine these operations as well. Here is a combination of + as well as - to give you a result.

[Video description begins] *In the input box of the next code cell, she types the following command: 27.5 + 9.4 -13. The number 7 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 23.9, displays below, after Out [7]:.* [Video description ends]

Obviously, these arithmetic operations will follow the BODMAS rule that we learned back in middle school. Brackets will be evaluated first, then division and multiplication, and then, addition and subtraction. Here is an example of how you would perform a multiplication operation in Python. Notice that we don't have a multiplication symbol on our keyboard which is why we use the * for multiplication. 2 * 6 gives us 12.

[Video description begins] *In the input box of the next code cell, she types the following command: 2 * 6. The number 8 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she the clicks the Shift and Enter keys and the output, 12, displays below, after Out [8]:.* [Video description ends]

Once again, you can mix integer values with floating points to perform your arithmetic operation. 10 * 5.5 will give us 55. Observe that the 55 is actually 55.0.

[Video description begins] *In the input box of the next code cell, she types the following command: 10 * 5.5. The number 9 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 55.0, displays below, after Out [9]:.* [Video description ends]

So, each time you have a floating point involved in your computation the result of that computation will always be a floating point value. If you have multiplication as well as addition operations, multiplication will be performed first based on the BODMAS rule. So, 2 * 3 * 1 gives us 6 + 2 gives us 8 and the results displayed here below your code cell is 8.

[Video description begins] *In the input box of the next code cell, she types the following command: 2 * 3 * 1 +2. The number 10 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 8, displays below, after Out [10]:.* [Video description ends]

Just like with ordinary math, if you want certain computations or operations to be performed first, you'll include them within brackets. So (1 + 2) will be computed first that will give us the result 3. Python will perform that computation before it performs the rest of the multiplication. So 2 * 3 * 3 gives us 18 and that is indeed the result.

[Video description begins] *In the input box of the next code cell, she types the following command: 2 * 3 * (1 + 2). The number 11 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 18, displays below, after Out [11]:.* [Video description ends]

You can also power of operations in Python and the way you indicate this is by using the double ** sign. This is 10 to the power of 2 which gives us 100.

[Video description begins] *In the input box of the next code cell, she types the following command: 10 ** 2. The number 12 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 100, displays below, after Out [12].* [Video description ends]

Let's try this once again so that you get familiar with it. 2 the power 4 which is indicated by the double ** sign will give us 16.

[Video description begins] *In the input box of the next code cell, she types the following command: 2 ** 4. The number13 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 16, displays below, after Out [13]:.* [Video description ends]

In the case of addition, subtraction, multiplication, and power of operations, if you specify only integers in the computation, the result will be an integer. If you specify a floating point, the result will be a floating point value. Division operations can be performed using the /.

Here the / indicates that you want to divide 10 by 5. 10 is the numerator 5 is the denominator. Observe that the result here is 2.0. In the case of the division operator, even though both of our input numbers are integers, the result is a floating point. If you're only interested in the integral result of the division, you can specify a different kind of operator and that is the //.

[Video description begins] *In the input box of the next code cell, she types the following command: 10 / 5. The number 14 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 2.0, displays below, after Out [14]:.* [Video description ends]

The double forward slash tells Python is that we're only interested in the quotient of the result. Remember, the result of a division is a quotient and a remainder. When you use the double forward slash, Python will only calculate the integral portion of the result, that is, the quotient and the result here is 2.

[Video description begins] *In the input box of the next code cell, she types the following command: 10 // 5. The number 15 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 2, displays below, after Out [15]:.* [Video description ends]

The difference between the double forward slash and the forward slash is more obvious when you perform a division operation such as 10.5 // 3. Now, you know from basic mathematics that the result here will be a floating point value. But we're interested only in the integer value of the result which is 3. The double forward slash has calculated just the quotient of the result. It has entirely ignored the fractional part or the remainder.

[Video description begins] *In the input box of the next code cell, she types the following command: 10.5 // 3. The number 16 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 3.0, displays below, after Out [16]:.* [Video description ends]

And that is why the result here is 3 rather than 3.5. And as you might imagine, just like you can calculate the quotient you can also ask Python to give you the remainder of a division operation. And this you do by using the % operation, also referred to as the modulo operator.

So, 50 % 4 here will divide 50 by 4 and give you the remainder as a result which is 2. Once you start programming a lot more, you'll find that this modulo operation is very useful for various numeric computations.

[Video description begins] *In the input box of the next code cell, she types the following command: 50 % 4. The number 17 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 2, displays below, after Out [17]:.* [Video description ends]

Let's try the modulo operation once again. This time we'll divide 50 % 5. This is an even division. There is no remainder and that is why the result here is 0. The remainder is 0.

[Video description begins] *In the input box of the next code cell, she types the following command: 50 % 5. The number 18 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 0, displays below, after Out [18]:.* [Video description ends]

Now that you have some basic practice using Python as a calculator you can perform more complex operations. Include the operations that you want performed first in brackets. Remember, the BODMAS rule has been

followed in every case. Here (4 + 7) will be computed first, we get the result 11. The result 11 will then be multiplied by 2 will give you 22 and you divide 22 by 4 to get the result. 22 divided by 4 will give you 5.5 and this 5.5 you will subtract from 40.5 to give you the result 35.

[Video description begins] *In the input box of the next code cell, she types the following command: 40.5 - 2 * (4 + 7) / 4. The number 19 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 35.0, displays below, after Out [19]:.* [Video description ends]

This is just a simple explanation of the BODMAS rule but it's important that you understand that Python works this way. Here is another computation but notice that there are outer brackets around the numerator. So, the entire numerator will be calculated first, and then, the result of that will be divided by 5. This is something that you can verify on your own and see if you get the result 37.

[Video description begins] *In the input box of the next code cell, she types the following command: (40.5 * 10 - 20 * (4 + 7)) / 5. The number 20 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 37.0, displays below, after Out [20].* [Video description ends]

Now it turns out that you can perform the multiplication operation where a string is involved as well. Here, the string is "Hello Oliver!" which is just a sequence of characters and which is enclosed in quotes, double quotes. In Python, you can use both the single quotes as well as the double quotes to enclose a string.

Here you multiplied the string "Hello Oliver!" by a number 3. What does this mean? Python basically interprets this to mean Hello Oliver 3 times. The result of this multiplication operation is this string concatenated 3 times and the result is also a string.

[Video description begins] *In the input box of the next code cell, she types the following command: "Hello Oliver!" * 3. The number 21 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and the output, 'Hello Oliver! Hello Oliver! Hello Oliver!', displays below, after Out [21]:.* [Video description ends]

The multiplication operation is the only one that works with strings. If you try "Hello Oliver!" + 3, this is not supported. And Python will display an error as you can see on screen. This error here can only concatenate str (not "int") to str. Means that the + operation can be used with two strings. And it'll simply concatenate the strings together; it cannot be used with a string and an integer.

[Video description begins] *In the input box of the next code cell, she types the following command: "Hello Oliver!" + 3. The number 22 now appears within the square brackets after the word 'In' in the command prompt next to this input box. Then, she clicks the Shift and Enter keys and an error message saying can only concatenate str (not "int") to str, displays.* [Video description ends]

Working with Python Built-in Functions

[Video description begins] *Topic title: Working with Built-in Functions. Your host for this session is Janani Ravi.* [Video description ends]

In this video, we'll introduce you to built-in functions in Python. Now, what exactly are functions? Functions you can think of as reusable bits of code and this can be code to perform a calculation, a useful action, anything.

For example, you could have a function to add two numbers, for instance, using the addition operation that we've seen earlier. Now, a good question to ask here would be where do these reusable bits of code come from?

[Video description begins] *A Jupyter notebook titled Built-InFunctionsInPython displays. A cell in the form of an empty input box displays with the command prompt: In [ ] : to the left of it. The cursor blinks in this input box.* [Video description ends]

Now, you, of course, can define your own function to perform your own custom action. But that's a little advance and will come later along as you study Python. Before that, you should know, that there are a number of built-in functions available in Python for you to use. And for this demo, we'll focus on these built-in functions. If you were to type in a string using double quotes into your code cell here on this Jupyter notebook, the Jupyter notebook will simply write out the string in your result.

[Video description begins] *In the input box of the first cell, she types the following command: "Hello world!". The number 1 now appears within the square brackets after the word 'In' in the command prompt next to the 1st input box. The following output displays below, after the word, 'Out' [1]: 'Hello world!'.* [Video description ends]

This is the default action of the Jupyter notebook where it simply echoes what you've typed in when it's a number or a string. If you're not working within Jupyter notebooks, or you want to explicitly, within a larger unit of code, ask Python to print something to screen, you will use the print function as you see here. Let's observe a bunch of details here. The name of the function is print and it's followed by brackets. This is how you know that it's a function.

A function has a name and it's invoked using brackets. Now, what we're passing in here, the string Hello Oliver!, is an input argument or an input parameter to the function. The input parameter is what the function needs, information that the function needs to perform its job.

Developers of the Python programming language have already written the code to print the information that you pass into the function out to screen. You are just invoking the print function and using their code. So, when we invoke the print function in this manner, it prints out to screen Hello Oliver!

[Video description begins] *In the input box of the next cell, she types the following command: print ("Hello Oliver!"). The number 2 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Hello Oliver!. She highlights the words, Hello Oliver! in the command to show that this is the argument for the print command.* [Video description ends]

Strings can be defined using double quotes in Python or single quotes. So, you can invoke the print function and pass in Hello Oliver! in single quotes as well and the same Hello Oliver! will be printed out to screen. Observe that, in both cases, when the print function printed out the string to screen, the quotes for the string were not included. Python only printed out the contents of the string, the content of what we passed into this print function.

[Video description begins] *In the input box of the next cell, she types the following command: print ('Hello Oliver!'). The number 3 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Hello Oliver!. Blue-colored arrows point to the output of the first command. The outputs of the next two commands are highlighted.* [Video description ends]

Now, you can't work with strings without the quotes. Let's say, I were to invoke the print function and pass in Hello Oliver!, without the enclosing single or double quotes, this is an error and Python will tell me that this is invalid syntax. Syntax basically refers to the structure of commands that Python understands.

[Video description begins] *In the input box of the next cell, she types the following command: print (Hello Oliver!). The number 4 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: SyntaxError: invalid syntax. A blue-colored cross symbol displays next to this print command. The error output is highlighted.* [Video description ends]

Python does not understand what you're trying to do because Hello Oliver is not in quotes. It doesn't know what it means; it considers it invalid. Let's say, you try and invoke print without the brackets but with only the double

quotes.

And in Python 3, this is an error. Programmers who are new to Python but might have encountered Python 2 earlier, will find this a little confusing because print with just the quotes worked in Python 2. That was a print command or a print statement which is different from a print function.

[Video description begins] *In the input box of the next cell, she types the following command: print Hello Oliver! The number 5 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hello Oliver!")? The error output is highlighted.* [Video description ends]

You don't need to worry about those details. You're working with Python 3. Always use the brackets. The print function to print stuff out to screen. Now that you've understood the print function,

[Video description begins] *In the input box of the next cell, she types the following command: print ("Hello Oliver!"). The number 6 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Hello Oliver!. She highlights the words, Hello Oliver! in the output.* [Video description ends]

we can move on to another built-in function in Python, the len function. The name of the function is len. It's invoked using brackets and the input argument that we pass into the len function is Hello Oliver!, the string. When you pass in a string to a len function,

[Video description begins] *In the input box of the next cell, she types the following command: len ("Hello Oliver!"). Blue-colored arrows point to the argument of this command.* [Video description ends]

the len function simply counts the number of characters in that string and, in our case, the number of characters is equal to 13. Observe that the code that was executed by the len function is different from the code executed for the print function. The len function calculates the length of what you pass in. The print function prints out what you pass in to screen.

[Video description begins] *The number 7 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [7]: 13. She highlights the output and blue-colored arrows point to the print and len functions above.* [Video description ends]

We'll now use the print function in a slightly different way, print 5 + 4. What does Python do? Python will first calculate the result here. 5 + 4 Python knows is 9 and it'll pass in 9 to the print function.

[Video description begins] *In the input box of the next cell, she types the following command: print (5 + 4). Blue-colored arrows point to the argument of this command and the command is highlighted.* [Video description ends]

The print function will receive the calculated result that is 9 and will print 9 out to screen.

[Video description begins] *The number 8 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [8]: 9. She highlights the output.* [Video description ends]

And this you'll see is true even for more complex calculations.

[Video description begins] *In the input box of the next cell, she types the following command: print (4.5 - 2 * (4 + 7)). She highlights the command.* [Video description ends]

Let's say you call print and, within the print brackets, you specify a complex calculation, the result of that calculation will be calculated first. That result is passed into the print function and the print function will simply

print the result out to screen.

[Video description begins] *The number 9 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [9]: -17.5. She highlights the output*. [Video description ends]

Python offers a whole number of such useful built-in functions such as the abs function here which calculates the absolute value of a number.

[Video description begins] *In the input box of the next cell, she types the following command: abs (17). She highlights the command. The number 10 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [10]: 17. She highlights the output*. [Video description ends]

The absolute value of 17 is 17. Let's use abs once again with -17. The absolute value of -17 is 17.

[Video description begins] *In the input box of the next cell, she types the following command: abs (-17). She highlights the command. The number 11 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [11]: 17. She highlights the output*. [Video description ends]

Just like with the print function, you can pass in the result of a computation to the absolute function as well.

[Video description begins] *In the input box of the next cell, she types the following command: abs (4.5 - 2 * (4 + 7)). She highlights the command*. [Video description ends]

So, Python will first compute 4.5- 2 * (4 + 7). The brackets will be evaluated first.

[Video description begins] *Blue-colored arrows point to (4 + 7) in the abs function*. [Video description ends]

Once the result of this entire computation has been calculated, that result is passed into the absolute function. And the absolute function will calculate the absolute value of that result which, in this case, happens to be 17.5.

[Video description begins] *The number 12 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [12]:17.5. She highlights the output*. [Video description ends]

The computed value of the result was -17.5. The absolute value of -17.5 is 17.5. Another built-in function that Python has is the type function.

[Video description begins] *She also highlights the output of the abs function in Line number 9 which is -17.5. In the input box of the next cell, she types the following command: type (17.5). She highlights the command*. [Video description ends]

All of these values that you're working with. We spoke about them as integers and floating points. The type function gives us the data type for a value. Here you can see that 17.5 is of type float.

[Video description begins] *The number 13 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [13]: float. She highlights the output. A blue-colored arrow points to 17.5 which is the argument she just passed to the type function*. [Video description ends]

Let's check out the type of just 17 without a point. You can see that it's of type int which is a short form for integer.

[Video description begins] *In the input box of the next cell, she types the following command: type (17). She highlights the command. The number 14 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [14]:: int. She highlights the output*. [Video description ends]

Let's check out the type of anything in quotes. Hello James! in quotes is of type string; str is its short form.

[Video description begins] *In the input box of the next cell, she types the following command: type ("Hello James!") She highlights the command. The number 15 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [15]: str. She highlights the output*. [Video description ends]

Let me point something out to you here and this is something you can try out on your own. Anything that you specify in quotes is of type string. Even if you pass a numeric value, whether it's an integer or a floating point in quotes, it is considered to be of type string. And this is something you can try out and see what result Python gives you. Let's move on. Let's use the round built-in function which rounds off a number to the nearest integer. Round on 17.7 gives you 18.

[Video description begins] *In the input box of the next cell, she types the following command: round (17.7) She highlights the command. The number 16 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [16]: 18. She highlights the output*. [Video description ends]

If you remember basic round operation from mathematics, it'll always round off to the nearest integer. So round on -17.5 will give you -18, that is, the nearest integer. Round on -17.4 will give you -17.

[Video description begins] *In the input box of the next cell, she types the following command: round (-17.5) She highlights the command. The number 17 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [16]: -18*. [Video description ends]

All of these functions work as you would expect them.

[Video description begins] *In the input box of the next cell, she types the following command: round (-17.5) She highlights the command. The number 18 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [18]: -17*. [Video description ends]

Python supports many built-in functions and an entire list is available here at this URL. Let's move on to exploring another useful built-in function called max which, as its name suggest, gives you the maximum value from a list of numbers that you can pass in.

Now, these can be any numeric values, integers, or floating points and notice that we can pass in any number of input arguments to the max function.

[Video description begins] *In the input box of the next cell, she types the following command: max (9, 5, 6, 12, 23, 18). He highlights the arguments of this function*. [Video description ends]

All of the functions that we saw earlier took in just one input argument in our example. Max finds the largest value amongst all of these numbers you've passed in. The largest value here is 23.

[Video description begins] *The number 19 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [19]: 23*. [Video description ends]

Just like the max function here we also have the min function that takes in a list of numbers. It can take any number of input arguments. The minimum value here in this list is the number 5 and that is what the min function returns.

[Video description begins] *In the input box of the next cell, she types the following command: min (9, 5, 6, 12, 23, 18). The number 20 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [20]: 5*. [Video description ends]

Introducing Python Variables to Store Values

[Video description begins] *Topic title: Introducing Variables to Store Values. Your host for this session is Janani Ravi*. [Video description ends]

Having understood mathematical operations and built-in functions, we can now move on to a very important concept in programming. Variables.

[Video description begins] *A Jupyter notebook titled CreatingVariables displays. A cell in the form of an empty input box displays with the command prompt: In [ ] : to the left of it. The cursor blinks in this input box*. [Video description ends]

So far we worked with all kinds of values such as integer values. This is the value 24000.

[Video description begins] *In the input box of the first cell, she types the following command: 24000. She highlights the command. The number 1 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [1]: 24000*. [Video description ends]

Now, this is a value, notice the terms that I'm using. You can subtract two values and get a result that is also a value. 24000 - 10000 will give you 14000.

[Video description begins] *In the input box of the next cell, she types the following command: 24000 - 10000. She highlights the command. The number 2 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [2]:14000. She highlights the output and command*. [Video description ends]

Let's perform another mathematical operation, 14000 - 5000. This gives you the value 9000.

[Video description begins] *In the input box of the next cell, she types the following command: 14000 - 5000. She highlights the command. The number 3 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [3]: 9000. She highlights the output*. [Video description ends]

Let's say you wanted to use this computed result 9000 in another place, you'd have to recompute it. 9000 isn't stored anywhere and that is where the importance of variables comes in. Variables in programming language are basically just containers which hold values.

And these values can by anything, numbers, strings, computed results, anything. Once a container holds a value, the value that that container holds can be updated later on in that program. Because it is a wrapper or holder of a value, that value can be changed. And this is how you would initialize a variable in Python.

[Video description begins] *In the input box of the next cell, she types the following command: salary = 24000. In next line, he enters salary*. [Video description ends]

Here, salary is a variable and notice how it is made up of alphanumeric characters, which are not in quotes. The = operator assigns a value to a variable. Here, the value 24000 is assigned to the container or variable salary.

[Video description begins] *A blue-colored arrow points to the = sign and 24000 is highlighted*. [Video description ends]

And if you type out salary, the value held in salary will be printed out to screen which is 24000. Within a Jupyter notebook cell,

[Video description begins] *The number 4 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [4]: 24000. She highlights the output*. [Video description ends]

when you type out a variable, the value held within that variable will be printed to screen. Let's give some meaning to the mathematical operations that we performed earlier. The salary is 24000, the house rent that you pay is 10000, and your food expenses are 5000.

[Video description begins] *In the input box of the next cell, she types the following command: house_rent = 10000 food_exp = 5000*. [Video description ends]

house_rent is a variable that stores the value 10000, food_exp is a variable that stores the value 5000. And if you simply type out house_rent within your code cell, the value held within it will be printed to screen.

[Video description begins] *In the input box of the next cell, she types the following command: house_rent. She highlights the command. The number 6 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [6]:10000*. [Video description ends]

Let's try the same thing with food_exp and you can see that what is displayed on screen is 5000. Now, that we have values stored within all of these containers,we can do interesting things.

[Video description begins] *In the input box of the next cell, she types the following command: food_exp. She highlights the command. The number 7 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [7]: 5000*. [Video description ends]

The total_expense that you incur is the sum of house_rent plus food_exp. Observe how you sum house_rent + food_exp. This will add up the values in those two variables and assign the result to the variable total_expense.

[Video description begins] *In the input box of the next cell, she types the following command: total_expense = house_rent + food_exp. total_expense. The values of house_rent and food_exp display on the right side of the screen for reference*. [Video description ends]

Python will perform the addition operation on the values within house_rent and food_exp and store the result in total_expense. When we type out total_expense, the result will be displayed. As you can see it's 15000.

[Video description begins] *The number 8 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [8]: 15000. She highlights the output*. [Video description ends]

So, if your total_expenses are $15,000, what are your savings? You can perform another calculation to get this information. You have your salary, you subtract your total expenses from the salary, and store the result in the savings variable.

[Video description begins] *In the input box of the next cell, she types the following command: savings = salary - total_ expense. savings. The value of salary displays on the right side of the screen for reference*. [Video description ends]

Simply type out savings within your code cell to see the value that it holds. You can see that total savings that we have is 9000.

[Video description begins] *The number 9 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [9]:: 9000. She highlights the output*. [Video description ends]

Let's now use the print built-in function that we've seen before, but this time we'll invoke the print function with two arguments. These arguments are separated by a comma. The first argument is a string which simply says my savings in a year is followed by a colon. That is the string in quotes. The second argument to the built-in function is our variable savings. So, the built-in function will take in the value of savings and print this out to screen in a concatenated manner.

[Video description begins] *In the input box of the next cell, she types the following command: print ("My savings in a year is: ", savings)*. [Video description ends]

My savings in a year is $9,000.

[Video description begins] *The number 10 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [10]:My savings in a year is: 9000. She highlights the output*. [Video description ends]

Let's work with some more variables so that we get used to this idea of containers for our values. course_fee is the variable which holds the value 299. num_of_months is a variable that holds the value 6. Variable names in Python can be made up of

[Video description begins] *In the input box of the next cell, she types the following command: course_fee = 299 num_of_months = 6*. [Video description ends]

uppercase and lowercase letters. So, A to Z, all uppercase, a to z, lowercase, digits 0 to 9, and the character _. No other characters are allowed in variable names. In addition, a variable name cannot start with a digit; it has to start with either underscore or a letter. Let's perform a few operations using these variables.

The total_fee is the course_fee for a month multiplied by num_of_months. So, instead of using values directly, you can use the variables, and print out the value of the result using the print function. total_fee is the variable on the left side of the assignment operator equal to which holds the result of this calculation which is 1794.

[Video description begins] *In the input box of the next cell, she types the following command: total_fee = course_fee * num_of_months. print (total_fee). The number 12 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [12]: 1794. She highlights the output*. [Video description ends]

We'll now create two more variables, num_1 and num_2, and assign the value 2 to both of these variables.

[Video description begins] *In the input box of the next cell, she types the following command: num_1 = 2 num_2 = 2. The number 13 now appears within the square brackets after the word 'In' in the command prompt next to this input box*. [Video description ends]

Let's invoke the print function. And this time we want to print out a computed result, num_1 + num_2. Python will first calculate the result which is 4 and the print function will print 4 out to screen.

[Video description begins] *In the input box of the next cell, she types the following command: print (num_ 1 + num_2). The number 14 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [14]: 4. She highlights the output*. [Video description ends]

The result that was calculated as an input to the print function wasn't really stored anywhere. If you want to store the result permanently for the duration of your program, you need to store it to a variable such as total here.

[Video description begins] *In the input box of the next cell, she types the following command: total = num_1 + num_2 . The number 15 now appears within the square brackets after the word 'In' in the command prompt next to this input box*. [Video description ends]

Now, if you print out the value of total, you can see that it contains 4, the result of num_1 + num_2.

[Video description begins] *In the input box of the next cell, she types the following command: total. The number 16 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [16]: 4. She highlights the output*. [Video description ends]

Python supports string concatenation using the plus operator. So, you can concatenate two strings, day and light, by simply using the plus operation and you get the result daylight.

[Video description begins] *In the input box of the next cell, she types the following command: "day" + "light". The number 17 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [17]: 'daylight'. She highlights the output*. [Video description ends]

But what if you tried to perform such an operation on two different types of values? So, you have num_1 which is assigned the number 2 and num_2 which is assigned the string 2. Remember, anything in quotes is a string even if it's a numeric value in quotes.

[Video description begins] *In the input box of the next cell, she types the following command: num_1 = 2 num _2 = "2"*. [Video description ends]

Now, if you try to perform the operation, num_1 + num_2, you will find that Python does not understand what it is you want to do.

[Video description begins] *In the input box of the next cell, she types the following command: num_1 + num_2. The number 19 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [19]: TypeError: unsupported operand type (s) for +: 'int' and 'str'*. [Video description ends]

This is an error; it is an unsupported operand. Plus is not supported between an integer value and a string value. How do we know the types are mismatched? Well, you can use the type function, type(num_1) is an integer; it's not in quotes.

[Video description begins] *In the input box of the next cell, she types the following command: type (num_1). The number 20 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [20]: int. She highlights the output*. [Video description ends]

And type(num_2) is of type str or string because this value is in quotes.

[Video description begins] *In the input box of the next cell, she types the following command: type (num_2). The number 21 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [21]: str. She highlights the output*. [Video description ends]

Working with Different Types of Variables in Python

[Video description begins] *Topic title: Working with Different Types of Variables. Your host for this session is Janani Ravi*. [Video description ends]

We've worked with integers, floating-point numbers, and strings. What we haven't seen yet are Boolean values. Boolean values refer to True and False, and True and False are both keywords in Python.

[Video description begins] *A Jupyter notebook titled CreatingVariables displays. In the input box of the first cell, she types the following command: x = True y = False. She highlights the command. The number 22 now appears within the square brackets after the word 'In' in the command prompt next to this input box*. [Video description ends]

Here the variable x, holds the Boolean value True and the variable y holds the Boolean value False. You can see that True and False are highlighted in green indicating they are keywords that Python understands.

I'll now invoke the print function here, which will introduce a number of new concepts, once again we've passed in two arguments to the print function. One is a string which says x and y is, followed by a comma, and then we have an operation x and y. This is a logical operation.

[Video description begins] *In the input box of the next cell she types the following command: print (' x and y is:' , x and y). She highlights the word, and, in the command*. [Video description ends]

If you remember logical operations from back in school, x and y will be True only if both x and y are True. x is True, y is False, the result of x and y will be False, and this is what you see here in the result, x and y is False.

[Video description begins] *The number 23 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: x and y is: False*. [Video description ends]

Python also supports other logical operators such as or. So we want to print the result of x or y, x or y will be True if either x or y is True. So the result here is True.

[Video description begins] *In the input box of the next cell, she types the following command: print (' x and y is:' , x or y). The number 24 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: x or y is: True. She highlights the output*. [Video description ends]

For the or logical operator, only one of the two values needs to be True. Here, x is True, which is why the result is True. Python also supports the negation or the not logical operator.

[Video description begins] *In the input box of the next cell she types the following command: print ('not x is: ', not x)*. [Video description ends]

We know that the value of x is True. So the not or negation of x is False, and that is what you see here in the result.

[Video description begins] *The number 25 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: not x is: False*. [Video description ends]

Let's do the same thing for y as well. We'll print out the not or negation of y, y is False. So the negation of not of y is True.

[Video description begins] *In the input box of the next cell she types the following command: print ('not y is: ', not y). The number 26 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: not y is: True*. [Video description ends]

Let's go back and work some more with string variables. Let's assign a new variable x with the value awesome, which is a string. The variable x here simply contains awesome.

[Video description begins] *In the input box of the next cell she types the following command: x = "awesome" x. The number 27 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: 'awesome'.* [Video description ends]

Remember you can concatenate strings using the plus operation. So we have two strings here, Python is in double quotes, that's simply a value. And we have the string in variable x, so x plus Python is will concatenate the two strings together, and we get the result Python is awesome.

[Video description begins] *In the input box of the next cell she types the following command: print ("Python is " + x). The number 28 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Python is awesome.* [Video description ends]

String concatenation is possible using the plus operator on two variables, we've initialized two strings here, x and y. x contains the value Python is and y contains the value awesome. Observe here that the variable x is a variable that we've used before, it earlier contained the value awesome.

This time we updated the value contained within this variable to be Python is. The whole idea of having a variable is that you can change the value that it holds based on your needs. Now, we are calculating x plus y and assigning the result to the variable z, and we're going to print out the value of z, which is simply Python is awesome.

[Video description begins] *In the input box of the next cell she types the following command: x = "Python is" y = "awesome" z = x +y print (z). The number 29 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Python is awesome.* [Video description ends]

If you remember from an earlier demo, you can multiply a string with an integer. So you have the string z, which contains the value Python is awesome. You can multiply it by four and store the result in a. 'a' here is simply a string. With the value Python is awesome, repeated four times thanks to our multiplication.

[Video description begins] *In the input box of the next cell she types the following command: a = z * 4 print (a). The number 30 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Python is awesome Python is awesome Python is awesome Python is awesome.* [Video description ends]

Let's invoke the len built in function on the variable x. The last assignment variable x was a string containing the value Python is followed by a space, so the total length of the string is 10.

[Video description begins] *In the input box of the next cell she types the following command: len (x). The number 31 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below after Out [31]: 10.* [Video description ends]

Observe how you can pass in variables as input arguments to any built-in function. The function will simply take the value within that variable and perform its actions and calculations on that value. len of z will compute the length of the string stored in z, which is Python is awesome with a length of 18.

[Video description begins] *In the input box of the next cell she types the following command: len (z). The number 32 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below after Out [32]: 18.* [Video description ends]

Assigning Values to Variables in Python

[Video description begins] *Topic title: Assigning Values to Variables. Your host for this session is Janani Ravi.* [Video description ends]

When you work with Python, you'll see that there are many different ways to assign values to variables.

[Video description begins] *A Jupyter notebook titled CreatingVariables displays. In the input box of the first cell, she types the following command: x = y = z = "Python" print (x) print(y) print(z). She highlights the command.* [Video description ends]

Observe the first line on screen here, x = y = z = "Python". Such a statement will be evaluated from right to left. So the rightmost value here, Python, will be assigned to the variable z. The value in variable z will be assigned to y.

The value in variable y will be assigned to x. Then finally, if you print out the values of x, y and z, you'll find that all of them have the string Python. This is a shortcut that you can use to assign multiple variables to the same value.

[Video description begins] *The number 33 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Python Python Python. She highlights the output.* [Video description ends]

We've already seen that you can concatenate strings using the plus arithmetic operator, b = x + y. So b is equal to 'PythonPython', since x and y both contain the string Python.

[Video description begins] *In the input box of the next cell, she types the following command: b = x + y b. The number 34 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: ' PythonPython'. She highlights the output.* [Video description ends]

I'll now introduce a new category of operators in Python, Comparison operators. Comparison operators can be used to compare two values. Result of a comparison is always a Boolean, True or False. The "==" is the equality comparison operator, then you have the "<", ">", the "<=", and the ">=".

Let's take a look at a rather confusing assignment here in Python, which uses the equality comparison operator. Remember that when you see assignments like this, evaluate this from the right to the left. Observe the == between y and z, this means that y is being checked to see whether it's equal to z.

If the values in y and z are exactly the same, the result will be true, and True will be assigned to x. If they are different, the result will be false, and False will be assigned to x, y is exactly equal to z, which is why x is True.

[Video description begins] *In the input box of the next cell, she types the following command: x = y == z print (x). The number 35 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: True. She highlights the output.* [Video description ends]

Python has first compared the value of y with the value of z, both contain the string Python. They are exactly the same, and the Boolean result which is True, y is indeed equal to z is assigned to the variable x. Here is another confusing assignment, a, b, c = 5, 3.2, "Hello".

Python will find the commas and assign values accordingly. 5 will be assigned to a, 3.2 will be assigned to b, and Hello will be assigned to c. And this is exactly what you get when you print out the values of a, b, and c.

[Video description begins] *In the input box of the next cell, she types the following command: a, b, c = 5, 3.2, "Hello" print (a) print (b) print (c). The number 36 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: 5 3.2 Hello. She highlights the output.* [Video description ends]

Let's study variable assignments in a little more detail. Here we have two variables, color_a and color_b assigned to Red and Blue respectively. And we print out the values of these variables. So Red Blue is printed out to screen.

[Video description begins] *In the input box of the next cell, she types the following command: color_a = 'Red' color_b = 'Blue' print (color_a, color_b). The number 37 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Red Blue. She highlights the output.* [Video description ends]

In Python, it's possible to assign a value to a variable using the value in another variable. For example, color_c = color_a, will take the value held in color_a and assign it to the color_c variable. And if you print out these values, you'll see that color_c contains the value Red, the same as in color_a.

[Video description begins] *In the input box of the next cell, she types the following command: color_c = color_a print (color_a, color_b, color_c). The number 38 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Red Blue Red. She highlights the output.* [Video description ends]

It's important here to note that color_c and color_a are both different containers containing the same value Red. Let's see what the implications of that are. Let's say you assign color_a to color_b. Color_a was initially Red, color_b was Blue. And now we get the result, color_a is now Blue but color_c continues to be Red. Updating the value within the color_a container did not affect the color_c container. Color_c contains the value Red as it did before.

[Video description begins] *In the input box of the next cell, she types the following command: color_a = color_b print (color_a, color_b, color_c). The number 39 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Blue Blue Red. She highlights the output.* [Video description ends]

Updating Variable Values in Python

[Video description begins] *Topic title: Updating Variable Values. Your host for this session is Janani Ravi.* [Video description ends]

In this demo, we'll further cement our understanding of how variables act as containers. And how we can change the values that live within these containers and the implications of this change. I'm first going to declare three numeric variables, n1, n2, and n3, and they contain either integers or floating-point numbers.

[Video description begins] *A Jupyter notebook titled MoreVariables displays. In the input box of the first cell, she types the following command: n1 = 34 n2 = 34.5 n3 = 100. She highlights the command.* [Video description ends]

I'm then going to move on and create two string variables, s1 and s2, the first of which contains the string Hello and the second contains World.

[Video description begins] *In the input box of the next cell, she types the following command: s1 = 'Hello' s2 = 'World'.* [Video description ends]

The variables that we have created hold the values that we've assigned those variables, and we can then use them in calculations. This is something that you're familiar with total_n = n1 + n2 + n3 will sum up the values in those variables and assign the result to a new variable total_n. And if you look at what total_n contains, it's 168.5, the sum of the three numeric values that we had set up earlier.

[Video description begins] *In the input box of the next cell, she types the following command: total_n = n1 + n2 + n3 total_n. The number 3 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [3]: : 168.5. She highlights the output*. [Video description ends]

Now the whole idea of the term variable is that the value that it holds can change. So you can reassign a value that is held within a variable. For example, when we say n1 = n1 + 1, Python will first evaluate the result n1 + 1. We know that n1 currently contains the value 34 it will add 1 to 34 get 35 and store this result in n1 itself. So if you print out the value of n1, you'll see that n1 now contains 35 it no longer contains the value 34.

[Video description begins] *In the input box of the next cell, she types the following command: n1 = n1 + 1 n1. The number 4 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [4]: 35. She highlights the output*. [Video description ends]

And such operations using variables in Python are extremely common, n2 = n2 + 10 will take the existing value of n2, add 10 to that, and then store the result in n2. So it'll take 34.5 + 10 = 44.5, and as you can see, n2 now holds the value 44.5. And really these operations are what make the variable get its name.

[Video description begins] *In the input box of the next cell, she types the following command: n2 = n2 + 10 n2. The number 5 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [5]: 44.5. She highlights the output. The values of n1, n2, and n3 display on the right for reference*. [Video description ends]

You can vary the value that are held within variables you can update these values. n3 - 50, n3 was initially 100, you subtract 50, you get 50, n3 now holds the value 50.

[Video description begins] *In the input box of the next cell, she types the following command: n3 = n3 - 50 n3. The number 6 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [6]: 50. She highlights the output*. [Video description ends]

Now, what if you perform another operation with n3, n3 = n3 multiplied by 50? In this computation, Python will use the last value of n3 that was stored within the variable. The last value stored was 50, so we get 50 multiplied by 50 the result 2500 is what is now will be stored in n3, and if you take a look at n3 you can see that its 2500.

[Video description begins] *In the input box of the next cell, she types the following command: n3 = n3 * 50 n3. The number 7 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [7]: 2500. She highlights the output*. [Video description ends]

Let's say we now perform one more computation with n3, n3 divided by 50. The last stored value in n3 was 2500, 2500 divided by 50 gives us 50, n3 once again contains the value 50.

[Video description begins] *In the input box of the next cell, she types the following command: n3 = n3/50 n3. The number 8 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [8]: 50. She highlights the output*. [Video description ends]

Let's now take a look at some other interesting syntax that Python supports. Let's see if you can figure out what Python will assign to n1 in this situation. The current value that is stored in the n1 variable is 35, we had initially initialized it to 34 and then incremented it by 1. So, Python will first evaluate the expression on the right-hand side of the assignment operator that is n1 + 1, and it'll get 36.

What you might have observed is that the += operator is not a simple assignment, and the fact is, it means addition plus assignment. It means, add what is on the right-hand side that is 36 to the current value of n1, which

so far is still 35. So it adds 36 to 35, and the result is stored in n1, that is what the equal to refers to in the +=. So the result is, n1 = 71.

[Video description begins] *In the input box of the next cell, she types the following command: n1 += n1 + 1 n1. The number 9 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [9]: 71. She highlights the output*. [Video description ends]

Evaluate the right-hand side get the result, add the result of the right-hand side to the value in the left-hand side and store the result in the left-hand side once again. Let's see a slightly simpler variation of the addition plus assignment to drive the point home. n2 += 10, the current value of n2 = 44.5, and on the right-hand side, we only have 10. So we take 10 add that to the current value of n2 and store the result in n2, so 10 + 44.5 = 54.5, n2 now has the value 54.5 stored within it.

[Video description begins] *In the input box of the next cell, she types the following command: n2 += 10 n2. The number 10 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [10]: 54.5. She highlights the output*. [Video description ends]

The use of the arithmetic operation in combination with the assignment operator extends to other arithmetic operators as well such as multiplication or division. n3 multiplied by equal to 50 takes the current value of n3 multiplies it by 50 and stores the result in n3, so n3 is now 2500. The value stored in n3 is now 50, let's try the division equal to operator.

[Video description begins] *In the input box of the next cell, she types the following command: n3 *= 50 n3. The number 11 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [11]: 2500. She highlights the output*. [Video description ends]

It takes the current value of n3, the last stored value in n3 is 2500, so 2500 divided by 50, store the result in n3 and n3 is now equal to 50 once again.

[Video description begins] *In the input box of the next cell, she types the following command: n3 /= 50 n3. The number 12 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [12]: 50.0. She highlights the output*. [Video description ends]

The combination of an arithmetic operator with an assignment operator is often used as a shortcut in Python and this is something you should be familiar with. We've seen that the arithmetic operator + can be used with strings as well. Here s1 + s2 will combine Hello + World and the result is a concatenated string HelloWorld, that is stored in the concat_string variable.

[Video description begins] *In the input box of the next cell, she types the following command: concat_string = s1 + s2 concat_string. The number 13 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [13]: 'HelloWorld'. She highlights the output. The values of s1 and s2 display on the right for reference*. [Video description ends]

We'll now perform a computation that changes the value that is stored in s1, s1 = s1 + 'Yoohoo' s1 was initially Hello. We concatenate Yoohoo to get HelloYoohoo, this is what is now stored in the s1 variable.

[Video description begins] *In the input box of the next cell, she types the following command: s1= s1 +'Yoohoo' s1. The number 14 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [14]: 'HelloYoohoo'*. [Video description ends]

Let's perform a similar computation with s2, s2 = s2 + Yippee, s2 was initially World, we concatenate Yippee to the end of s2 and s2 is now currently WorldYippee.

[Video description begins] *In the input box of the next cell, she types the following command: s2= s2 +'Yippee' s2. The number 15 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [15]: 'HelloYipee'.* [Video description ends]

We'll now assign the current value of s1 to a variable called new_s1, this is an entirely new container, an entirely new variable. new_s1 contains the current value of s1, which is HelloYoohoo.

[Video description begins] *In the input box of the next cell, she types the following command: new_s1= s1 new_s1. The number 16 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [16]: 'HelloYoohoo'.* [Video description ends]

Now you've assigned new_s1 to the original value of s1, and let's say you change the value that is stored in the s1 variable, you set it to some other string. Now, s1 is equal to some other value, now when you take a look at the contents of new_s1 you want to see what values held within it.

[Video description begins] *In the input box of the next cell, she types the following command: s1= 'Some other value' s1. The number 17 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [17]: 'Some other value'.* [Video description ends]

You'll find that it remains unchanged it continues to be HelloYoohoo. This is the value that we had assigned to new_s1 so even if we had changed s1. The variable that was created using the last value of s1 continues to hold this last value HelloYoohoo.

[Video description begins] *In the input box of the next cell, she types the following command: new_s1. The number 18 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [18]: 'HelloYoohoo'.* [Video description ends]

In order to understand this point more clearly, let's see this with another variable, the numeric variable n3, we assign the current value of n3 to a new variable, some_n. So some_n currently has the value 50.

[Video description begins] *In the input box of the next cell, she types the following command: some_n = n3 some_n. The number 19 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [19]: 50. The value of n displays on the right for reference.* [Video description ends]

We'll now perform an arithmetic computation which changes the value held in n3. We multiply n3 by 1000 and store the result in n3 itself, so n3 has a value of 50,000.

[Video description begins] *In the input box of the next cell, she types the following command: n3*= 1000 n3. The number 20 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [20]: 50000.0.* [Video description ends]

Now some_n, which we assigned the value of n3 earlier, continues to have the value 50, because that was the value of n3 at the time of assignment. So even though we've changed the value of n3 some_n continues to be 50, this is important for you to remember.

Once we've created a new container, it continues to hold the value it was assigned until you explicitly change that particular variable or that particular container.

[Video description begins] *In the input box of the next cell, she types the following command: some_n. The number 21 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [21]: 50.0.* [Video description ends]

Working with Python Simple Data Types

[Video description begins] *Topic title: Working with Simple Dat Types. Your host for this session is Janani Ravi*. [Video description ends]

In all of the demos so far, we worked with a number of different types of values, strings, floating-points, integers. Let's take a look at the different data types available in Python.

[Video description begins] *A Jupyter notebook titled TheDataTypesInPython displays. A cell in the form of an empty input box displays with the command prompt: In [ ] : to the left of it. The cursor blinks in this input box*. [Video description ends]

These are the built-in data types available by default. You can always create your own data types using classes, but that's far more advanced. We'll come to that when we are ready. We've seen earlier that the type function will give us the data type for a particular value. Type of 23 will tell us that this is of type int, which is a short form for integer.

[Video description begins] *In the input box of the first cell, she types the following command: type(23). She highlights the command. The number 1 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [1]: int*. [Video description ends]

If you pass in a decimal to the type function, the type function will correctly interpret this of type float. This is a short form for floating point numbers.

[Video description begins] *In the input box of the next cell, she types the following command: type(25.5). She highlights the command. The number 2 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [2]: float*. [Video description ends]

The input argument to a type function can be a variable as well. Here we have the variable named x, which is assigned the value 25 which is an integer. There are a number of details to observe here, observe that we invoke the print function with four arguments. The first is a string data type of, followed by a comma, then x, then is, that is a string in single quotes, and then the type of x.

And here is the most interesting bit, we've passed the result of the function type into the print function. So Python will first calculate the result of the type function and pass that result into the print function and the print function will print out the data type of 25 is int. Now, because x is a variable and not a value, it says class int because int is one of the built-in classes in Python.

[Video description begins] *In the input box of the next cell, she types the following command: x =25 print ("Data type of" , x, 'is', type (x)). She highlights the command. The number 3 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Data Type of 25 is <class 'int'>*. [Video description ends]

This additional information, the fact that int is a Python class is visible to us here because we passed in the result of type to the print function and the print function displayed this information. All of the other built-in types in Python are also classes, for example, the variable y here is assigned to 8.2. And when you print out the type of y as we do in this print function, you can see that it is of class float.

[Video description begins] *In the input box of the next cell, she types the following command: y =8.2 print ("Data type of" , y, 'is', type (y)). She highlights the command. The number 4 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Data Type of 8.2 is <class 'float'>*. [Video description ends]

Once again, you don't need to know what exactly classes are. You can just say this is of type float. Now, Python supports not just integers and floating point numeric types. It also supports complex numbers. For example, the variable z here is assigned the value 5j, which indicates that z is a complex number. So if you print out the type of z, you can see that its type is complex. It is of class complex.

[Video description begins] *In the input box of the next cell, she types the following command: z =5j print ("Data type of" , z, 'is', type (z)). She highlights the command. The number 5 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: Data Type of 5j is <class 'complex'>*. [Video description ends]

Python also supports arithmetic operations on complex numbers. You may not use it very often but you should know that it's possible. 5j * 5j, will give you -25 + 0j. This is a real number -25. If you can't remember complex numbers from your high school math, you can just forget about it. But you should know that working with complex numbers is possible in Python, in case you ever have to do so.

[Video description begins] *In the input box of the next cell, she types the following command: print (5j * 5j). She highlights the command. The number 6 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: (-25 + 0j)*. [Video description ends]

We work with boolean_values before, boolean_values can take on two forms, True or False. The type of a boolean_value is simply bool.

[Video description begins] *In the input box of the next cell, she types the following command: boolean_value = True type (boolean_value). She highlights the command. The number 7 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [7]: bool*. [Video description ends]

We've also worked with string values before. Here, the string commando is assigned to str_value, that is the name of the variable, and the type of the variable will be str. This is of class string.

[Video description begins] *In the input box of the next cell, she types the following command: str_value = "commando" type (str_value). She highlights the command. The number 8 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after the word, 'Out' [8]: str.* [Video description ends]

Python also gives you the flexibility to specify numbers in the binary octal or hexadecimal format. These are different bases for numbers, when you specify numbers in the binary format, you will prefix the binary representation of your number with 0b. Here is the binary number 111, notice the 0b prefix. This is the number 7, and that is what the print function realizes and prints to screen.

[Video description begins] *In the input box of the next cell, she types the following command: print (0b111). She highlights the command. The number 9 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: 7*. [Video description ends]

Here is a binary number assigned to the value variable, 0b101. And 101, as you know, is the number 5, and that's what is printed to screen.

[Video description begins] *In the input box of the next cell, she types the following command: value = 0b101 value. She highlights the command. The number 10 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [10]: 5*. [Video description ends]

If you pass in a binary number to the type function, you'll see that the binary number is nothing but an integer.

[Video description begins] *In the input box of the next cell, she types the following command: type (0b101). She highlights the command. The number 11 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below, after Out [11]: int.* [Video description ends]

Creating Single-line and Multi-line Strings in Python

[Video description begins] *Topic title: Creating Single-line and Multi-line Strings. Your host for this session is Janani Ravi.* [Video description ends]

There are lots of little details to understand when we work with strings in Python, and that is what we'll cover here in this demo.

[Video description begins] *A Jupyter notebook titled UseOfQuotesInStringAndSpanMultipleLines displays.* [Video description ends]

A string is a sequence of characters specified within double quotes or single quotes. Here is a single quoted string,

[Video description begins] *In the input box of the first cell, she types the following command: print ('I am a single quoted string'). The number 1 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: I am a single quoted string.* [Video description ends]

the very same string can be specified within double quotes as well. And that is also a string and behaves in exactly the same way as a single quoted string.

[Video description begins] *In the input box of the next cell, she types the following command: print ("I am a double quoted string"). The number 2 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: I am a single quoted string.* [Video description ends]

You can use three different kind of quotes to create strings in Python. So you can have a double quotes, you can see the word python is enclosed in double quotes. You can use single quotes, the word is, is enclosed in single quotes, or you can specify a string using three double quotes on either side of the string. You can see the specification for the word awesome, which is also a string in Python. And you can concatenate all of these strings together to get python is awesome.

[Video description begins] *In the input box of the next cell, she types the following command: print ("python " + 'is ' + " " " awesome " " "). The number 3 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: python is awesome.* [Video description ends]

We've already seen that a string can be multiplied by an integer. And the result will be a string with the original string repeated that many number of times. Here the word python has been repeated three times thanks to our multiplication by three.

[Video description begins] *In the input box of the next cell, she types the following command: print (' python ' * 3). The number 4 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: python python python.* [Video description ends]

Other arithmetic operations don't make sense in the context of strings, such as subtraction. You can't really subtract an integer from a string, and Python will accordingly throw an error.

[Video description begins] *In the input box of the next cell, she types the following command: print (' python ' - 3). The number 5 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: TypeError: unsupported operand type(s) for -: 'str' and 'int'*. [Video description ends]

Python has this special function called the input function. Which allows Python to accept input from the user, and this input is in the form of a string. The input function that you see here invoked on screen will accept user input and assign that user input to the answer variable. You will then use the string in the answer variable and print some information out to screen, You had answer, That sounds delicious!

[Video description begins] *In the input box of the next cell, she types the following command: question = "What did you have for lunch?" answer = input ( ) print ("You had " + answer + " ! " + " That sounds delicious ! ")*. [Video description ends]

When you hit Shift+Enter and execute this bit of code, you'll find that an input text box comes up on screen within your Jupyter Notebook. Waiting for your input. You can type in something meaningful into this text box, such as pasta.

[Video description begins] *An input box displays below the command and she types the word, pasta, in it*. [Video description ends]

The string pasta will be assigned to the answer variable, and you get this result printed to screen. You had pasta, That sounds delicious! This is from our print statement.

[Video description begins] *The following output displays below: You had pasta! That sounds delicious!* [Video description ends]

Now why does it matter that Python accepts strings using both single as well as double quotes? Let's see an example where the quotes actually matter. Let's say that your string itself included a single quote, such as the word I'm, here in this example.

After the I we have a quote. And Python, because we've enclosed the entire string in single quotes, assumes that the second quote after I closes the string. So when you execute this bit of code, Python will tell you the syntax is invalid. Because it doesn't understand what the rest of the characters after the I represent.

[Video description begins] *In the input box of the next cell, she types the following command: print (' I 'm learning python'). The number 7 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: SyntaxError: invalid syntax*. [Video description ends]

If your string itself includes a single quote, you should enclose that string using double quotes. Such as the example you see here on screen. I'm learning python is enclosed within double quotes, and Python correctly interprets the single quote within the string as a part of the string.

[Video description begins] *In the input box of the next cell, she types the following command: print (" I 'm learning python"). The number 8 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: I'm learning python*. [Video description ends]

Similarly, if your string itself includes double quotes, you should enclose the string as a whole in single quotes. John said, Hello there! Hello there has double quotes around it, that is a part of our string, which is why the entire string has been enclosed in single quotes. And when you execute this code, you'll see that Python correctly interprets Hello there within double quotes as a part of the string.

[Video description begins] *In the input box of the next cell, she types the following command: print (' John said "Hello there!"'). The number 9 now appears within the square brackets after the word 'In' in the command*

prompt next to this input box. The following output displays below: John said "Hello there!". [Video description ends]

If your string itself includes a single quote and you have to enclose your string in single quotes for some reason. You need to escape the single quote within your string. Notice the backslash character just before the single quote before m. This backslash is termed as an escape character. And it tells Python whatever character comes after this backslash, accept that character as is, as a part of the string. Don't apply any special meaning to that character. So the single quote just before m will not be treated as the termination of a string, it'll just be treated as a part of the string.

[Video description begins] *In the input box of the next cell, she types the following command: print (' I\'m learning python'). The number 10 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: I'm learning python*. [Video description ends]

Escape characters work well, but it makes code hard to read, which is why they're not often preferred. Here is an example where we use double quotes within a string which is enclosed in double quotes. So the quotes within the string have been escaped using the backlash character. Thanks to the backslash, which escapes the quotes within the string, the quotes are interpreted correctly by our Python interpreter.

[Video description begins] *In the input box of the next cell, she types the following command: print (" John said \ "Hello there!\" "). The number 11 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: John said "Hello there!"*. [Video description ends]

Here are some more slightly more complex examples using quotes and strings. So the entire string is enclosed in single quotes, which means we can use double quotes within the string without escaping the double quotes. Any single quote within the string has to be escaped as we've done. If you want to include a new line within a string,

[Video description begins] *In the input box of the next cell, she types the following command: print (' I \'m learning "python"'). The number 12 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: I'm learning "python"*. [Video description ends]

you can do this using the \n special character. \n will tell Python to add a new line between Hello John! and How are you? And this is what you see here in the result.

[Video description begins] *In the input box of the next cell, she types the following command: print (" Hello John! \n How are you?"). The number 13 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following two lines of output display below: Line 1: Hello John! Line 2: How are you?*. [Video description ends]

You can have multiple new lines within a string if you specify multiple \n's in your string. Here is a very long string that spans three lines thanks to the multiple \n's we have in there.

[Video description begins] *In the input box of the next cell, she types the following command: print (" This is a very long line \n but we can have it span \n multiple lines. "). The number 14 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following three lines of output display below: Line 1: his is a very long line Line 2: but we can have it span Line 3: multiple lines*. [Video description ends]

Using special characters within string often reduces the readability of our code. Which is why if you have multi-line strings, you should use the triple quote. Here is a single line string enclosed within triple quotes. Now, this is no different than a string specified within single quotes or double quotes.

[Video description begins] *In the input box of the next cell, she types the following command: print (" " "I am a triple quoted string " " "). The number 15 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: I am a triple quoted string*. [Video description ends]

Let's say you have a string that includes a new line. You want the string to span multiple lines. Instead of using the \n, you could simply specify the string using three single quotes. You can see that the resultant string spans two lines, there is a new line right after the word long.

[Video description begins] *In the input box of the next cell, she types the following command: print (' ' 'This is a very long string in python ' ''). The number 16 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following two lines of output display below: Line 1: This is a very long Line 2: string in python*. [Video description ends]

So for strings that span multiple lines, use the triple quotes. You can use three single quotes or three double quotes, they work exactly the same way. So rather than using the new line character, the best practice in Python is to use the triple quote for multi-line strings.

[Video description begins] *In the input box of the next cell, she types the following command: print (" " "This is a very long string in python " " "). The number 17 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following two lines of output display below: Line 1: This is a very long Line 2: string in python*. [Video description ends]

You can see a number of different examples here. All of these are strings with multiple new line characters. All of these work, and they can be specified using triple quotes.

[Video description begins] *In the input box of the next cell, she types the following command: print (" " "This is also a very long string but we can print it across multiple lines " "). The number 18 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following three lines of output display below: Line 1: This is also a very long string Line 2: but we can print it Line 3: across multiple lines*. [Video description ends]

The triple quotes here are a must. Try specifying a multi-line string, which has a new line character within it using double quotes. And you'll see that Python will immediately throw an error. It's not able to parse the new line within double quotes.

[Video description begins] *In the input box of the next cell, she types the following command: print (" This is a very long string, in python "). The number 19 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: SyntaxError: EOL while scanning string literal*. [Video description ends]

Formatting Operations with Strings in Python

[Video description begins] *Topic title: Formatting Operations with Strings. Your host for this session is Janani Ravi*. [Video description ends]

Let's see some ways in which you can include a variable to be part of your string.

[Video description begins] *A Jupyter notebook titled UseOfQuotesInStringAndSpanMultipleLines displays*. [Video description ends]

Here we have a variable called my_var, which is set to the string value barks. You can perform string concatenation manually. So here we are concatenating the string "A dog" with the string in my_var, plus the exclamation point to get A dog barks!

[Video description begins] *In the input box of the active cell, she types the following command: my_var = 'barks' print ('A dog ' + my_var + '!'). The number 22 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: A dog barks!.* [Video description ends]

This is manual concatenation using the arithmetic plus operator, this kind of statement is hard to read. If you specify your string using the f prefix in Python, Python will interpret anything within curly braces inside your string as an expression. The f prefix here basically tells Python that the my_var within curly braces is actually a variable or an expression and should be interpreted as such.

When you take a look at the result here, you'll see that the value within curly braces has been interpreted as an expression. We get the value of the my_var variable which is the word barks and that is what has been inserted into the resultant string.

[Video description begins] *In the input box of the next cell, she types the following command: print (f 'A dog { my_var } ! '). The number 23 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: A dog barks!.* [Video description ends]

If you don't specify the f prefix for the string, the value my_var within curly braces will be interpreted literally as my_var within curly braces. Without the f prefix, this is not interpreted as an expression.

[Video description begins] *In the input box of the next cell, she types the following command: print ('A dog { my_var } ! '). The number 24 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: A dog { my_var }!.* [Video description ends]

The f prefix can be used with single-quoted, double-quoted and even triple-quoted strings. And anything within curly braces in that string will be interpreted as an expression.

[Video description begins] *In the input box of the next cell, she types the following command: print (f ' ''A dog { my_var } ! ' ' '). The number 25 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: A dog barks!.* [Video description ends]

Another longer way to format your string to include expressions is to call the .format function and pass in the variables you want to include within the string. Here, the value within the variable my_var, that is the string barks, will be inserted into the position of the curly braces in our string to get A dog barks!

[Video description begins] *In the input box of the next cell, she types the following command: print ('A dog { } ! ' . format (my_var)). The number 26 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: A dog barks!.* [Video description ends]

Let's take a look at a few other special characters that can be included within a string. The \t stands for a tab, this is how you include a tab within a string. Notice that there is a tab space between the word asked and How are you?

[Video description begins] *In the input box of the next cell, she types the following command: print (" John asked \t 'How are you? ' "). The number 27 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: John asked 'How are you?'. A single tab space, denoted by a blue-colored arrow, exists after the word asked.* [Video description ends]

Let's see another example with a tab, Hello \t World. So between the words Hello and World in the result, there are two spaces and one tab space.

[Video description begins] *In the input box of the next cell, she types the following command: print ("Hello \t World"). The number 28 now appears within the square brackets after the word 'In' in the command prompt next*

to this input box. The following output displays below: Hello World. A single tab space, denoted by a blue arrow, exists between the two words. [Video description ends]

If the backslash character is used to escape codes and for special characters, how do we include a backslash within our string? Well, we escape it using another backslash. Yes, this is confusing, but this is the standard way most programming languages do this. So if you want a backslash within your string, you simply specified two backslashes. The first backslash, will tell Python to interpret the second backslash, literally. And you can see that the resulting string has just a single backslash after colon and after Users.

[Video description begins] *In the input box of the next cell, she types the following command: print (' C: \\ Users \\ Q '). The number 29 now appears within the square brackets after the word 'In' in the command prompt next to this input box. The following output displays below: C: \Users\Q.* [Video description ends]

Exercise: Python Jupyter Notebooks, Functions, & Variables

[Video description begins] *Topic title: Exercise: Jupyter Notebooks, Functions, & Variables. Your host for this session is Janani Ravi.* [Video description ends]

Here, are some questions for you to ponder on, in this exercise. You will first explain the features and benefits of using Jupyter notebooks to write Python code. You will then try and recall and list four built-in functions available in Python and how these functions are used. You will then try and define what a variable is in Python and show code examples which use variables.

And finally, we'll discuss strings. You'll explain the different ways to create strings in Python. And when you would choose to use one method to create strings over another. As you ponder over these questions and think of answers, I recommend that you pause this video for a little bit before moving on to look at some sample answers.

[Video description begins] *Solution* [Video description ends]

When working with Python, Jupyter notebooks are one of the most popular code editors out there. And a large proportion of the Python community swears by Jupyter notebooks. Jupyter notebooks are a browser-based shell where you can write Python code. It's not a desktop application, though you do need to install Jupyter notebooks which you can do via the Anaconda distribution. The biggest reason why Jupyter is so powerful and so popular is because of the interactive environment that it offers within the browser.

You can write your Python code within special cells called code cells and execute it right away. You simply hit Shift + Enter, your code will be executed, and results of the code execution will be displayed right below the code cell. Whether it's an error or the execution result, all of these are available to you within the same browser window. Jupyter notebooks are extremely easy to install. They are part of the Anaconda distribution that we used in this course. They're intuitive to work with and allows you as a learning programmer to focus on the code rather than on the intricacies or nuances of dealing with the integrated development environment or the IDE.

Your answer regarding the features and benefits of using Jupyter notebooks might be a little different, but these are the key important points. Now let's move on and talk about the built-in functions available in Python. Once again, the built-in functions that you thought of might be a little different. The first built-in function that you probably use in Python is the print function. The print function takes in any number of input arguments and these input arguments can be of any data type, and prints those out to screen in the form of a string.

Any function in Python including the built-in functions are invoked using parentheses and you specify the input arguments within these parentheses. The next built-in function, another one that is useful, is the len function. The len function is typically used to find the length of a sequence. This can be a list or even a string. Another widely used and useful built-in function is the type function.

This takes in a value or a variable and displays the data type for that variable or value. Now data types can be floats, integers, strings, or complex data types as well and type works with all of these. Another example of a built-in function used in Python is the round function. The round function works with numeric values such as integers and floating point numbers. It takes in a numeric value and rounds it to the next highest integer, based on what's in the position after the decimal point.

When you're writing code when you're writing programs, you have several values that you might want to work with. Values can be stored in containers and it is these containers that are referred to as variables in Python. The values that you store within Python variables can be of any data type. They can be simple data types such as Booleans, that is True or False values. Numeric values, just integers and floats or even strings. Variables can also store values of complex data types which you may not be familiar with yet, such as lists and dictionaries.

Variables are a fundamental building block in Python or in fact in any programming language. Instead of using values directly, you can have a meaningful variable which represents what that value is. Salary, cost, revenue, profit, all of these are valid variable needs. The advantage of using a variable rather than a value directly is that if you want to change the value stored in a variable, you simply make the change in one place where you assign a value to the variable.

Changing the value stored in the variable will change all locations where the variable is referenced. Let's see some code examples that use variables. Here we are defining two variables num_1 and num_2, and assigning them the values 1 and 10 respectively. We can now use these variables in arithmetic calculations.

So we store num_1 + num_2 the result is stored in the total variable. Total will contain the value 11. You can also use the equal to or the assignment statement to update the values stored in a variable. Here I'm updating the values stored in num_1, I'm adding 2 to it so num_1 at the end of this code execution will contain the value 12.

[Video description begins] *The following four lines of code display: Line 1: num_1 = 1 Line 2: num_2= 10 Line 3: total= num_1 + num_2 Line 4: num_1= num_1 + 2*. [Video description ends]

Let's now see an example of variables that store strings. Here, the variable s stores the string Hello specified in double quotes and w stores the string World specified once again in double quotes. You can now perform operations on these string variables as well, the string final will now hold the sum of s + w.

[Video description begins] *The following three lines of code display: Line 1: s = "Hello" Line 2: w = "World" Line 3: final = s + w.* [Video description ends]

It'll hold the string Hello World. Let's see one more example of initializing a floating point variable f = 2.3. Now I'm going to subtract 1 from f which I do using the minus equal to assignment operator. This will subtract 1 from f and the result 1.3 will be stored once again in the variable f.

[Video description begins] *The following two lines of code display: Line 1: f = 2.3 Line 2: f - =1*. [Video description ends]

Let's now move on to answering the next question on strings in Python. Strings in Python can be defined within single quotes. The string that you see here on screen is a valid string in single-quotes. Strings in Python can also be defined within double-quotes, and strings in Python can be defined within triple-quotes as well.

All of these three methods used to define strings are perfectly valid in Python. But in some circumstances you might want to choose one over the other. Let's say you have a string with double-quotes within it, such as around the quotes word. In that case you will want to define your entire string within single-quotes that it's easier to read. Let's say your string itself has a single-quote within it, such as the word I'm, which has a single-quote.

In that case, you might want to define the entire string within double-quotes, making it easier to read. And finally, triple-quoted strings are typically used for multi-line strings. If you are specifying a really long string that spans multiple lines, you'll enclose the string in triple quotes.