Python: The Big Picture

by Jason Olson

Course Overview

Course Overview

Hi everyone. My name is Jason Olson, and welcome to my course, Python: The Big Picture. I am currently a senior software engineer at Concur Technologies where I build and maintain distributed systems every day with a focus on high availability and fault tolerance. Python is a storied programming language that continues to grow in popularity. It's been used everywhere from web development to data science, and even for special effects in movies. Python is a very powerful programming language that is well suited for many different types of tasks. This course is a quick introduction to Python, and no prior experience with Python is expected nor required. Some of the major topics that we will cover include what Python is, what makes Python different from other programming languages, why developers should care about Python, seeing Python in action using different technologies, and tools you will need to know to develop using Python. By the end of this course, you will have the necessary knowledge to start experimenting with Python and to dive deeper into the areas where Python is being used. I hope you'll join me on this journey to learn more about Python with this Python: The Big Picture course, here at Pluralsight.

What Is Python?

Introduction

Hi. This is Jason Olson, and welcome to this Pluralsight course that aims to help you understand the big picture of the Python programming language. Python is a programming language that is rapidly growing in popularity and being used in many different ways and industries. In this module, we will discuss what Python is to help provide the foundation for the rest of this course. In other modules, we will cover the many different places Python is being used, how you can get started using Python, and common tools and resources that will help you on your Python journey. Before you know it, you will be a Pythonista in form and spirit. So with that said, let's get started.

Getting the Most from This Course

First, prerequisites. This course is a basic course introducing the Python programming language. No prior Python knowledge or experience is required. You are likely to get more value from this course if you do have some experience writing code, though. The primary audience for this course are those that are involved in the day to day of creating and shipping software. This includes not only developers and testers, but also development managers. To get the most from this course, feel free to join the discussion. If you have any questions, you can find the Discussion tab on the Pluralsight page for this course. I am also available on Twitter @jolson88. I would love to hear about your experiences with this course, so let's get started.

What Is Python?

The first step in our journey is to understand what Python is. So what is Python? At its core, Python is a high-level programming language. You won't have to deal with low-level assembly instructions, machine-specific details, or other minute things. With Python, you get to enjoy all the benefits that using any high-level programming language brings you. Python is a language with a dynamic type system. This means that any type checks in the programming language happen at runtime instead of compile time. Programmers that prefer a dynamically typed language often discuss how dynamic languages are usually simpler, have smaller source footprints due to the lack of type annotation spread throughout the code, they're quicker to read due to less type clutter, and with the lack of a compiler phase, they can lead to a quicker turnaround when making software changes. Python and the culture of Python programmers place a huge emphasis on readability, so after you know Python, it can be a very pleasant experience reading and learning other Python code. Fundamentally, Python is usually used as an interpreted language. This provides very powerful prototyping and live coding opportunities, and can also aid in live

debugging efforts. It does mean the code can be slower than compiled code, but engineering is always a game of tradeoffs. Python is also a multi-paradigm programming language. Python directly supports both object-oriented and structure-oriented programming. It also embraces many functional programming concepts, especially when working with groups of things. Overall, Python is a very powerful language that can be used in many different ways and in many different areas. So where did Python come from, and what kind of popular usage has it seen so far? Python's first major release was in 1991. Guido van Rossum, its creator and community-titled benevolent dictator for life, started the project in 1989. He decided to write an interpreter for a new scripting language he had been thinking about, one that was a descendant of the ABC programming language and one that would appeal to UNIX and C hackers. Python 2.0 was released on October 16, 2000, and had many major new features. It included both a cycle detecting garbage collector and support for Unicode. With this release, the development process was also changed and became more transparent and community backed through the use of Python Enhancement Proposals, also known as PEPs. Early in its development process, Python 3.0 was commonly referred to as Python 3000, or Py3K. It was a major backward incompatible release and was initially released on December 3, 2008, after a long period of testing. Now, many of its major features have been back ported to the backwards compatible Python 2.6 and 2.7 version series. Along the way, there have also been many cool usages of Python as well. In 1999, Python was used as a scripting language to help make special effects by industrial light and magic on the movie Star Wars: The Phantom Menace. Also in 1999, Anaconda was introduced as the new installer technology for Red Hat Linux. Anaconda itself was created with Python. In 2003, the modeling application, Blender 2.26, was released, its first free public release, even though it was created in the mid to late '90s. Blender uses Python as a scripting language for building blender add-ons and to otherwise extend blender functionality. Then in 2011, Google released Google App Engine. Python was one of the first major languages supported for building applications on Google App Engine. Over its many years of usage, as you can see, Python has grown increasingly powerful, very capable, and used in many different areas.

Why Should Developers Care?

So why should a developer care about all of this and learn a new programming language like Python? One factor is Python's growing popularity. When many developers think of popular programming languages, they often think of Java, C#, C++, JavaScript, and perhaps even PHP. Python doesn't usually come to mind, but according to some recent Stack Overflow analysis, Python is quickly overtaking other languages like Java, C#, C++, and PHP on Stack Overflow

question views. And according to PYPL, the Popularity of Programming Languages index, Python is the second most popular programming language, growing as much as 4% in popularity over the last year alone. It doesn't stop there, though. Python has the highest spectrum ranking according to a recent IEEE Spectrum report, and the TIOBE index has Python in the top five of programming languages. There's no longer any denying that Python is a popular programming language, but there are other reasons to care about Python as well. Some of the largest benefits actually come from the philosophy of Python that the community and culture is built upon. Many of these come directly from PEP 20, often referred to as the Zen of Python. The following five items are just a sampling of the principles outlined in it. Beautiful is better than ugly. Sometimes we have a tendency as developers to get lost in the search for clever solutions. Unfortunately, clever solutions tend to also be harder to read and understand, and in many people's opinion is just flat out uglier. In Python, there's a seeking for beauty that happens, treating programming much more as an art than a science. Explicit is better than implicit. Implicit assumptions or features are problematic. They decrease the ability to read code, and it becomes more difficult for other developers to understand what you were trying to do. Simple is better than complex. As developers, we can often be our own worst enemies. Let's admit it. We sometimes go for the more clever or smart solution. That's the easy thing to do, though. Simple is hard. Distilling a problem down to its core essence and finding a simple solution is incredibly difficult to do. We often fall short of this goal as developers, though, incurring tech debt in the process. As Alan Kay says, "Perspective is worth 80 IQ points." Complex is better than complicated. It's not always possible to avoid complex solutions. Sometimes a problem itself is complex. This is referred to as necessary complexity. But when a solution needs to be complex, that doesn't mean it needs to be complicated. Complication often stems from one piece of code, dealing with many different things at once. In Clojure, a different programming language, Rich Hickey has termed this as complecting many different things together, they become intertwined and braided together. This makes the code more difficult to work with, more difficult to refactor, to test, and even harder to understand in the first place. To understand a single piece of code, we have to keep many different concepts in mind all at the same time. The last one we'll discuss here is that readability counts. You'll notice in many of the preceding four points that a common underlying tie is the focus on readability and being able to understand code. Most of the cost of software comes in maintaining the software. Any time we can make code easier to read and understand, we are helping address this most expensive cost of developing and delivering software. We not only save pain for other developers, but we also help our future selves when we need to come back and work with the code we have already written. A focus on readability also helps make the code easier to discuss with other developers. To quote Tony Hoare, "There are two ways to write code:

write code so simple there are obviously no bugs in it, or write code so complex that there are no obvious bugs in it." By focusing on readability, we are striving to write code that is so simple there are obviously no bugs in it. And these are just a highlight of the top values. There are more that come along with this. Given the popularity of Python, the power of its community and culture, and the foundational principles that Python code is built on, I think it's a very powerful and capable high-level programming language that every developer should take seriously and have fun playing around with.

What Makes Python Different?

So what makes Python different than other programming languages? Why might you want to learn about and use Python when compared to the many other choices available today? The way many developers view it, there are four major areas that make Python different than many other programming languages out there. First is an emphasis on an extensible design. Python was designed to be highly extensible as a programming language. This means that many pieces of functionality can be added on top of the core Python language, while also providing a first-class development experience. This allows the core language to remain small and powerful. In this core, Python rejects exuberant syntax and excessive syntactic sugar, focusing instead on a less cluttered language, which contributes to its readability. Python can also be directly embedded into an application to provide a programmable interface, as we will see with applications like Blender. Python's development is conducted largely through the Python Enhancement Proposal process, also known as PEP. The PEP process is the primary mechanism for proposing major new features, for collecting community inputs on an issue, and for documenting the design decisions that have gone into Python. Outstanding PEPs are reviewed and commented on by both the Python community and by Guido van Rossum. This has enabled Python to evolve as a shared language and evolve to users' changing needs over the years. An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python, and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard foo and bar. You can also see this reflected in the many creative coding libraries that are out there available in Python today. All of this comes together in a very embracing culture and community built around Python usage. Key beliefs around readability and fun have combined with strong beliefs in avoiding premature optimization and having a form of idiomatic Python code to provide a robust culture that continues to grow Python and embrace newcomers. It's always a great time to become a new Python developer.

Summary

To recap, in this module, we took a look at what Python is, Python's history, and why developers should care about it. In the next module, we'll take a look at the many different areas and ways that Python is being used today.

When and Where Is Python Being Used?

Introduction

Welcome to another module in this course on Python: The Big Picture. This is Jason Olson. In this module, we will continue on our Python journey. So far we've discussed what Python is and a bit of its history. In this module, we're going to discover the many different ways Python is being used today. We will also see it in action in many of these areas as well. This module aims to serve as a bit of inspiration by showing you the sheer variety of areas and ways Python is being used. Investing in learning Python is never a poor investment, so let's dive right in and get started.

Linux Scripting and Administration

One of the first areas we'll discuss where Python is being used is in the world of Linux scripting and administration. When we are scripting to administer a machine, we obviously have a machine that lies at the heart of it. There are lots of things we may want to interact with and do with that machine as well. We routinely work with folders and files on the machine. We might be managing a series of log files or simply examining and monitoring the file system. Of course, one common reason to interact with the file system is when dealing with the configuration of apps and services. We may want to monitor the processes running on a system, keeping an eye on any potential runaway processes that need to be kept in check. Or, we may want to launch new processes as part of our scripting process. We might want to deploy a new application or deploy a version of an existing app onto that machine. We may even want to remove older applications that are no longer needed. Finally, we could even be running scripts to execute various test suites. These could be used to make sure the machine itself is healthy to validate existing apps and services on the machine, or we might be intentionally breaking apps and services to test how resilient our various systems are in the presence of failures. There are many different things you

may want to use scripting for, and with Python as a vehicle, the main limit is our imagination. We have full access to the wide variety of Python libraries that are already out there or that we can create from scratch. So let's dive in and take a quick look at two simple scenarios you may encounter in this area, working with files and working with processes. Let's start by working with files. We have a JSON file that contains some data. Let's take a look. As you can see, it's a JSON object that contains a name and a list of hobbies. We'll start by creating a new Python file called input.py. The first thing we do is to import the JSON module, as this will allow us to easily work with the JSON data we're going to suck in from the input JSON file. To get the data, we use the open function and give it the name of our file and specify we're opening it up in read-only mode. By using the with statement, the file handle will automatically be closed when the code block exits. Once we have the file open, all we need to do is use json.load to load the data from our input file into a Python object we can work with. For now, we'll just use the name from the object to say hello to. We save the file and change back to our terminal. We execute the file using Python 3, and we can see our message printed to the command line like we expect. So let's change our code to also write to a file to see the other side of the equation. Similar to reading a file, we just use the open function to open a file to write to. This time we use w, for write mode, and use the with statement again to automatically close the file handle when we're done with it. We start by writing out the name and prepare to print out the name of the hobbies listed in the JSON object. To iterate over the hobbies, we'll just use a for statement over the array in our JSON object. For each hobby, we simply call the write method on our file object to write a string to the file. A quick save of the file, and then we can execute it again using Python 3. It runs successfully with no errors. To see the output, let's print the contents of our output.txt file to the screen. We can see the format we expected printed out. As we saw, working with files is very straightforward and simple. For the last part of the demo, let's switch over and take a brief look at one way we can work with processes. First, we'll create a new file called process.py. In this file, we're going to list all the running processes on the system as a child process to our running code, and then we'll use a pipe to communicate with that child process so we can retrieve the output of the process we executed. The first thing we need to do is import the subprocess module that enables us to easily work with child processes in Python. All we're going to do is use the Popen function, for process open, in the subprocess module to execute our ps command as a child process. We specify that standard out of our child process is a pipe that we can then communicate directly with. After the subprocess is executed, its results can be found in the variable pl. The variable pl, in this case, will be a byte array containing the output of the command we executed. So we will decode the output as a Unicode string and just print it to the console. After we save the file and switch back to the terminal, we can execute it using Python 3. As we can see, it lists out all the running

processes in the system as a child process and printed its contents out to the console, just like we expected.

Web Development

Another area that Python is seeing growing use in is the area of web development. There are many different buckets or styles of web development. You might be developing just an HTTP or REST API, and hence not dealing with front-end user interfaces, just back-end service code. Or, you might be building a full stack web application that touches both the client and the server. You might even be working with an out-of-the-box content management system application, an ERP application, or an application of a different flavor. No matter which it is, though, Python is likely being used already. If you are building APIs, you can find existing popular Python frameworks like Flask, Bottle, Pyramid, or a myriad of others. Same goes with full-stack web development. You have Django, TurboGears, web2py, and many others. And even for full self-contained web applications, you can find apps like Plone, django CMS, Mezzanine, and many more. No matter what type of web application you're developing, Python is being used out there today. So let's take a quick peek of this in action. Let's take a look at two short Hello World examples in Python, one using Flask for API development and one using Django for a full-stack web application. First, let's create a quick web API using the Flask microframework for Python. We start by creating a Python file called service.py. We need to import Flask from the Flask module in order to create our own Flask web API. We create a new Flask app by creating an instance of Flask, giving it the name of the application. For this demo, we will just give it the name of our Python file. Now that we have our app, we can start creating a route for our web API. Let's go ahead and create a route at the root of our URL structure. To define the functionality of this route, we just define a function immediately below the route declaration. For this demo, we simply return Hello World from the function. Finally, we save the file and switch back to the terminal to run our new Flask web API. We specify an environment variable which tells Flask what file constitutes the starting point of our Flask app, and then tell Flask to run the application using the command flask run. We can see it spin up and tell us it's available to be used on port 5000. Now if we make an HTTP request to the root URL on port 5000 of the local machine, we see our Hello World message returned like we expected it to be. Though we are using cURL here, you could do the same thing using your browser or any other tool you wish to use. The Flask microframework isn't limited to just the most simple route declarations either. When developing a proper web API, you will need to parameterize the URL structure as well. For example, let's create a user route that includes the given username in the URL. We can parameterize this quite easily in Flask using brackets. Then

we get the user name as a normal function parameter in our route function and can echo the value back to show it in action. We save the file, and then we'll rerun our service using Flask. Now when we retrieve a user URL, we can see the username echoed back to us as we expected it to be. While being a very small and self-contained microframework, Flask gives you all the power you need to create a full-blown web API using Python. But how about creating a full-stack application in Python? Well, that's easily done in Django, just one of many frameworks available to us. To create a new Django application after we have installed Django, we just run the django-admin command and tell it to start a new project. We will call it mysite in this case. Once we navigate into the folder it created, we can see all the files that it created for us by default. These are primarily around setting up URLs that will render appropriate views for different parts of our application. To run the application, we use the manage.py file that was generated by Django for us. Just execute manage.py with Python 3 and give it the command runserver. The server spins up, and we can see that it's running on port 8000 by default. When we open up our browser to port 8000 on the local machine, we are greeted with the default new project Django landing page. Now we can begin creating new views and everything else you would expect when building out a new website application. We won't do more Django work right now, though, as there's an entire course on Django and Python that you can find, right here on Pluralsight, that will give you everything you need if you're interested in seeing more of Django in action. So with this tutorial at its end, you can see that any web development you may want to do you can absolutely do in Python. Python is more than up to the task and is rising in popularity in the web development space as well.

Application Scripting

Another area we will look into in this module is the topic of application scripting. So what is application scripting? One way to think of application scripting is to think about how we typically think of an application itself. We tend to think of an application as a standalone package whose functionality is all self-contained. As a user, we typically don't think of an application as something that can be easily extended by us. The application is exactly what we're given. No more, no less. But what if a user or developer could write their own code to respond to events that are raised within the application? Or going even further, what if a developer could write their own user interface elements, new screens, new user flows, or something else, and integrate it directly into the application itself? Not many applications written today take advantage of how powerful a concept this is. Games are one type of application that this is more common in, but it's less common to find this in your typical application outside of that. But this is exactly what application

scripting aims to do, and is one area Python can be used to a very powerful degree. You can find it in 2D and 3D modeling software, image manipulation software in the form of different imaging processing algorithms, photo taking applications in the form of fun overlays over pictures, and other types of applications as well. Let's take a quick look at this in action. In this demo, we're going to use Blender and experience using Python to interact with Blender in an interactive way. Welcome to Blender 3D, a free, open source 3D modelling tool. Blender supports Python as a scripting language and is really fun to play around with. We've just opened a brand-new project, which has the standard cube and camera. To get started, we will change the Bottom view from the standard Timeline view to the built-in Python Console view. You'll notice this is a full-blown Python REPL. We can investigate all objects within our scene using the Blender data exposed to Python via bpy.data.objects. The output is not directly helpful yet, but it does show that we have objects in our scene. Let's convert it to a list first so we can see what the contents of the objects property is. You can see three objects in our scene, a camera, our cube, and a lamp that lights the scene. Let's play around with the cube. The first thing we do is bind a local variable to the cube object. We can then investigate and interact with the cube object using Python code. For example, we can print out the current location of the cube. It shows it's located at the coordinates 0 0 0, as we can confirm in our 3D view as well. Let's change the cube's location directly in the 3D view by moving it around the scene. Now when we check the location of our cube variable again, we see the change we just made is reflected. It's important to realize that dragging the cube around in the 3D view and changing it in code are effectively the exact same operation. To check that out, let's import the mathutils module, which will make working with vectors a bit easier. Now, let's change the location of our cube using code. We will change its location by 1 in each dimension. Keep an eye on the view of the cube. As soon as we enter the code and press Enter, we see the cube's location immediately change. As a person learning Python, you can imagine how much fun this is to have a REPL where you can directly interact with and change a 3D modeling scene. It's a fun playground for sure. Blender scripting interface gives us access to practically everything we could possibly want access to within the scene. Hopefully you can see how powerful this could be when implementing more advanced scenarios. And now, you get an idea of the kind of power that can be made available to developers by using Python as a scripting language in your applications.

Data Science

Data science is a term that is growing in popularity with each passing week and day. Data science is a very large field, though. Two common areas that tend to be wrapped up in the data science

field are those of big data and machine learning, so let's take a look at both of them. First up, big data. What is big data? Well, as developers we're used to thinking of data in terms of kilobytes, megabytes, and perhaps gigabytes. But data is growing rapidly. More and more systems are needing engineers to start thinking in the terms of terabytes, petabytes, and even in some cases exabytes. Yes, exabytes. The amount of data being generated each and every day is growing rapidly and shows no signs of stopping. As of 2012, 2.5 EB of new data were generated every single day on the internet. To put that in perspective, that's 2.5, followed by 18 zeros, in bytes, per day. Datasets have grown so large that existing data processing systems are simply unable to deal with them. New systems are needed. This is where the growing popularity of the Hadoop, MapReduce, and its ilk is coming from. What used to be a problem faced primarily by large companies like Google or Microsoft is now starting to be faced by smaller enterprise businesses as well. The more data we have available to us, the more we can spot new business trends, find correlations to help prevent new diseases, combat crime, and much more. And yes, Python is a very popular programming language used in this space. So how about machine learning? Machine learning isn't directly related to big data; however, the more data we have available to us, the more powerful machine learning becomes. So what is machine learning? Well, from a 30,000-foot level overview, today we are surrounded by data everywhere. It comes in all forms, shapes, and sizes. It might come from large text files. It might be generated by the actions of our users. It may even be large amounts of metadata available in pictures and videos. The many relationships that might exist between all this data is much more complex than any single person or group of people could find by themselves. So instead, algorithms are written in such a way that a computer can process all the data and find its own connections between the many different sources of data. The algorithm then outputs its results in a way that can be digested by humans or built in to applications. The real power, though, comes from the fact that it's very common for these algorithms to feed the results back into themselves, learning how we can analyze the data even better for more accurate results. This is where you may hear the terms guided or directed learning, neural networks, and the like. This grew out of the field of artificial intelligence and has now found its way into the cracks of many different enterprises. You can see it in determining whether a given piece of email is spam or not. Or you might find it being used for the purpose of network intrusion detection, using the vast amount of network traffic to detect what's abnormal traffic or not. It's used for optical character recognition to convert images of text into text data that a computer can understand and process. It's also used for computer vision problems like face detection, object tracking, and motion tracking. And this is all just the tip of the iceberg as well. And what's exciting is that Python is being used in this field more and more every single day.

Summary

So, in this module, we learned about some of the different places Python is being used and how. We learned about its use in system administration, in web development, machine learning, big data, and application scripting just to start. As you've seen, Python is a very powerful language and can be used in most situations a programmer is likely to come across. In other modules, we discussed what Python is and a bit of its history. So, now in future modules, we'll prepare to take our first steps with Python and we'll take a look at other steps we can take to continue on our Python journey after this course is over.

First Steps with Python

Introduction

Welcome to another module in this course on Python: The Big Picture. This is Jason Olson. Other modules so far have discussed what Python is and showed where and when it is being used. In this module, we're going to take our first steps with Python. First, we will briefly discuss the minimal core basics that you need to know in order to work with Python code. Next, we will talk about different ways and resources you can use to learn more about Python. Lastly, we will discuss how to work with code from other people. These three areas will give you enough information to start working with Python, which we will do in another module. So let's get started.

The Basics

The basics. Let's take a look at some of the core basics that will help you when working with Python code, interacting with the Python community, or learning more about Python. Perhaps your first source of interaction with Python is going to be the main python.org website. This website contains an absolute wealth of information about Python and the Python community. You can learn more about Python as a programming language, including reading some tutorials on getting started using Python. You can find information on how to download and install Python for any platform you're running. You can also find detailed documentation on various Python topics and programming language features. You can even find a bunch of information on the community surrounding Python, including success stories from people who have embraced Python, the latest news in the Python world, and upcoming Python events. As you can see, python.org is a great one-stop shop for Python information if you are just getting started with it. Installing Python is

quite straightforward. Running Windows or Mac, you can find links to the different types of installers, archive packages, or offline docs. But installing Python can be even easier than this. Both Windows and Mac have popular command line installer helpers like Choco or Homebrew that makes installing software very simple and straightforward. If you are running Windows, simply do a choco install python command. And voila! You will have Python installed on your machine and ready to use. On Mac, it's the same with Home brew. Just run brew install python3, and you're good to go. We will talk more in a different module on the differences between Python 2 and Python 3, so don't let that confuse you for now. Of course on Linux, you can use the built-in package manager for your distribution as you already use when installing other software. So how do you actually work with Python code? What are the tools you will find yourself interacting with most of the time? The first tool you will interact with is the Python executable itself. This is the main executable that you use to execute or run a Python file. If you start it without specifying a specific Python file, it will launch you into a rudimentary REPL that you can use to play around with Python code. If you aren't familiar with REPLs, REPL stands for read-eval-print loop. It lets you type in a Python statement, it evaluates it, prints the result, and loops around waiting for you to type in another Python statement. When learning to play around with libraries, this interactive development experience is very powerful and fun to use. Another tool you will be using is a tool called pip. One of the believed definitions of pip is that it is a recursive acronym that stands for pip installs packages. You can use pip to install third-party libraries into your project very easily. It is similar to other package managers that exist for other platforms, like npm for Node.js, Gems for Ruby, NuGet for .NET, and many others. The last two you will find yourself hopefully using is an executable called IPython. While the Python executable provides a rudimentary REPL to use, IPython is a very robust, interactive shell that provides the same features and much, much more. It is a tool often overlooked by new Python developers. We take a look at getting started using these tools in a different module when taking our next steps on our Python journey.

Learning and Documentation

So now that we know how to install Python and can install it, how can we go out and learn more about Python? Obviously, watching the rest of this course is a start, but let's discuss other resources that are out there for you to learn more about Python after this course. We've already briefly mentioned the Getting Started guide that can be found on the official python.org website. This is a great guide to work through. If you are the type of person that prefers to learn by doing and working through a guide, make sure to check this one out. One shouldn't overlook the value for learning in the official docs themselves, either. In the official docs available at docs.python.org,

there are several different areas you can learn more about. You can read about what's new in each successive release. You can read tutorials and how-tos to learn more about Python. You can read the reference docs to learn more about specific features in various official libraries or the language itself. You can also learn more about working with Python modules, including installing them or making your own to distribute to others. Heck, you could even read up on extending and embedding Python as a runtime, or even using the Python/C API if you're wanting to play around with C or C++ code. The docs are rather all encompassing as you can see, and there's more that we haven't seen yet. If you click the What's new area of a Python release, you will be taken to the release notes for that Python version. One thing I really love about these release notes is that they include links to all the individual PEPs, or Python Enhancement Proposals, for all new features and enhancements in that release. By clicking on any single PEP in this list, you are taken directly to that PEP. Here you can read the proposal itself, including overviews of the feature, the spec of the feature, or even concerns about the feature. This provides very transparent insight into the new features that are released with each successive Python release, and it emphasizes some of the strength that comes from having a community-driven development process as a programming language. Lastly, for learning and documentation, you of course have Pluralsight courses. There are a large number of courses relating to Python that are available right here at Pluralsight. If you are new to Python, two courses in particular that would be great to work through are Python: Getting Started and Python Fundamentals. As mentioned, there are a large number of other Python courses that cover a wide array of topics. You can find topics related to system administration and scripting, web development, machine learning and data science, desktop development, embedded scripting, and much, much more. There are a lot of great resources out there to learn Python. Hopefully these become just the first steps in a much longer Python journey for you.

Working with the Code of Others

To build nearly any Python app or service of value, you're going to need to work with the code of others at some point in time, so let's take a look at how we can do that. Obviously, when working with other's code, we need to know that others are, in fact, writing Python code that we can use. How might we go about discovering that though? In the Python world, there is the Python Package Index, or PyPI. This is where people often publish their Python code to so it can be used by others. Finally, we pull Python code from PyPI into our own project using pip, which we discussed before. So what is pip, and how do we use it? Pip provides us several areas of functionality to work with for us using the code of others. Obviously, we can install and uninstall

Python packages. This is the main purpose for a package manager in the first place. But is there anything else? Why, yes, there is. Pip helps us by not only letting us track our own dependencies, but also by installing and uninstalling the dependencies of the Python packages we are installing into our project. We can also track a group of Python packages as a package group. This allows us to install, uninstall, and otherwise manage the package group as a whole rather than interacting with each individual Python package by itself. Pip can also help us make sure we are using the right versions of the packages and other dependencies that our application or service needs. This helps prevent us from breaking our software by unknowingly trying to use a different version of a package than we actually support. Pip comes installed by default with many Python distributions and installs, so you won't often need to install pip yourself. But if you do, there is a bootstrapping script hosted on the pip project that can be downloaded and executed with Python that will go ahead and install pip on your machine. Once this installation process finishes, you are able to go ahead and use pip. Installing packages with pip is incredibly straightforward. It's as simple as executing pip install followed by the package you want to install. Then that package is ready to be used in your project. As we've already mentioned in this module, PyPI, the Python Package Index, is where you can go to see all the various packages that are available to be used. You can find PyPI at pypi.python.org. PyPI is easy to browse using your web browser, but it does also support a JSON, XML-RPC, and HTTP interfaces to work with if you need to work with it programmatically. You can also find some links on this main page on how to upload your own packages to PyPI, to submit bug reports, and much more. So, in this module, we learned about some of the basics of working with Python code, online resources that enable us to learn more about Python, and the main ways to work with other people's Python code. In other modules, we discussed what Python is and showed where and when it is being used. So in the next module, we'll take a look at what other steps we can take to continue on our Python journey after this course is over and see Python in action again.

Continuing Your Python Journey

Introduction

Welcome to another module in this course on Python: The Big Picture. This is Jason Olson. In this module, we will continue on our Python journey. Other modules so far have discussed what

Python is, showed where and when it is being used, and even taking our first steps working with Python code. In this module, we're going to take a look at what we can do to continue our Python journey and see Python in action again. First, we will briefly discuss the difference between Python 2 and Python 3. When working with Python code, you will need to understand the difference. Next, we will talk about executing Python code and take a look at it in action. Lastly, we will discuss some potential next steps on our Python journey. So with all that covered, let's get started.

Python 2 vs. Python 3

Perhaps one of the most confusing aspects of Python development to new Python programmers is understanding the difference between Python 2 and Python 3, or even why a difference exists in the first place. Python 2 and Python 3. As we covered in a different module, Python 3 was first introduced in 2008. The major split stems from the fact that Python 3 is intentionally not backwards compatible. This enabled many powerful features to be added to the Python programming language and runtime without needing to support existing Python 2 code. By default, when working with Python 2, we will be using the executables pip, python, and ipython. However, when working with Python 3, we will typically be using these same executables, but with the version reflected in the executable name, pip3, python3, and ipython3. Sometimes you will even see pip2, python2, or ipython2 used to make the difference more explicit when using Python 2 with some distributions of Python. Depending on your installation or distribution of Python 3, it may not have had a Python 3 compatible version of IPython installed with it. If not, you can easily install one using pip3. This is great and all, but when should you use Python 2 and when should you use Python 3? The easiest way to think of it is that, well, you should default to always using Python 3 when possible. Python 3 is the version of choice. Python 2 is officially legacy code. Python 2 is officially considered in its end of life now. There are a couple of reasons you might need to use Python 2, though. If you don't have any control over the environment you were deploying your code to, Python 2 might be your only option at runtime. This could be because of a cloud platform that you're running on or limitations in an IT commissioned server that your code runs on. You might also have required libraries your code needs to use that aren't compatible yet with Python 3. As Python 3 was first released in 2008, there have been many years for lots of packages and software to get upgraded to Python 3 now, so this situation is becoming less and less likely with each passing month, but it's a situation to still be aware of. So to recap, your default first choice should always be Python 3. Only fall back to using Python 2 if it's largely out of your control and Python 3 is not an option.

Executing Python Code

Let's take a look at actually executing some Python code ourselves. There are three primary ways that we execute Python code. One of the first ways we learn to execute Python code is by using the interpreter. To use the interpreter, we simply execute a Python file using the python executable. This is perhaps the simplest and most common way to get started executing Python code, as we'll see in a moment. Another way to execute Python code is to use the REPL, the read-evaluate-print loop. We do this by inputting Python code directly into an interactive prompt or by loading up a Python file as a module into the REPL and calling its functions directly from the REPL. This can be done using either the Python executable or using the more purpose-built IPython. The final way to execute Python code is to run it natively. When run natively, you don't need to worry about a specific version of the Python runtime, or even any Python runtime at all, being installed on the system. This is done using a compile and run mechanism. We can do this via tools like py2exe, PyInstaller, or a number of other options that are available to us. So, let's step into the command line and take a look at executing Python code in action ourselves. Let's start by creating a new Python file called hello.world.py. I'll just use vi for now, but this could be any text editor or IDE of your choosing. For now, we will just print Hello, World and save the file. After switching to a new terminal window, we then can simply execute our Python file using Python 3. As we can see, it prints out Hello, World, like we expected it to. Now we'll edit our hello.world.py file to have a function instead that we will call later. We save the file and switch back to our terminal window. As we talked about before, the main Python executable also has a REPL mode as well. We start that by launching Python 3 with no parameters. Now we are in the REPL. It's waiting for us to give it a Python command to execute. If we call the print function, we will see it print out the string to the screen. The REPL supports inputting any valid Python statement. If the statement is incomplete, like when using an if statement, the REPL will show you that it's waiting for more input. Once the statement is complete, it is then executed and the result printed out like normal. Returning to our Hello World example, we come to have another way to execute our own code. We can have our code in a separate Python file and import it into our REPL. Now that we have Hello World imported, we can call the sayHello function that we created within the file. After doing so, we see the message printed out as we expected. One catch to be aware of is that any changes you make aren't automatically reloaded, though. Let's switch windows back to our Python file and change the message it prints. After saving the file, if we execute the function again, we will see that no change has taken place. This is because the code from before was already loaded. In this case, we need to explicitly tell the REPL to reload the file. We do this by importing the importlib module. With importlib imported, we call its reload

function, passing in our hello_world module. This then reloads the module from disk. Now we can see that calling the sayHello function reflects the changes we made by printing out our new message. Some quick notes, though. If you're using a version of Python 3 less than 3.4, you will need to import the imp module instead of importlib. As of 3.4 though, imp has been deprecated in favor of importlib. If you are using Python 2 instead of Python 3, you call the built-in reload function. You can see from our error here that the built-in reload function is not supported anymore in Python 3, though. Reloading every time can obviously get very annoying, very quickly, and slow down your prototyping process. Well, this is just one example of many where IPython can streamline your experience when compared to the default Python REPL. So let's launch IPython and see it in action. We're going to load an IPython extension called autoreload that will take care of reloading any modules for us. We tell autoreload to reload a module any time Python code within it is executed and the file has been changed. Like before, we're going to import our hello_world module. For now, we are going to use a different form of the import statement that allows us to import just the sayHello function specifically. You will notice that even without the autoreload so far that IPython is a better experience. With IPython, we are getting syntax highlighting, and we will also get auto indenting and other nice features. It's a good Python editor built in directly to the REPL experience. Finally, we execute our function and see that it prints the latest version of the text we have. To see autoreload in action, we switch back to our hello_world file, change the text that it's printing, and save the file. Now when we switch back to our REPL and immediately execute the sayHello function again, we can see the latest code is being used. We didn't need to reload the import ourselves. This is just a brief example of how you can execute Python code, but should give you a good start on messing around with Python yourself.

Summary

This course is coming to a wrap now. What are some next steps you can take after you are finished with this course? As we've talked about, there are some other great Python courses right here on Pluralsight that can serve as good next steps for you. We've already briefly mentioned the courses on Getting Started with Python and Python Fundamentals, but it doesn't stop there. You can find a course on using Python for Linux System Administrators, a course on doing Full Stack Web Development with Python, and even a course on Understanding Machine Learning with Python. And this is just the tip of the iceberg as well. Of course, you can also work through the many guides, tutorials, and documentation that are out there that we've discussed in previous videos. It's really your choice. So, in this course, we discussed what Python is and showed where and when it is being used. We've learned about some of the basics of working with Python code,

online resources that enable us to learn more about Python, and the main ways to work with other people's Python code. Finally, we discussed ways after this course is over to continue on our Python journey. I sincerely hope you've enjoyed this Python Big Picture course. Thank you for taking the time to watch and learn with me. I wish you all the best in your future adventures with Python!

Course author

Jason Olson

Jason Olson is a software engineer passionate about distributed computing and cloud-based technology. He is a full stack developer at Concur, and formerly a Technical Evangelist and Program Manager...

Course info

Level	Beginner
Rating	★★★★★ (698)
My rating	★★★★★
Duration	1h 5m
Released	26 Jan 2018

Share course