**IOC Topic 7A.2 Part 2 – Advanced Python**

Transcript & Notes

Author: Dr. Robert Lyon

Contact: robert.lyon@edgehill.ac.uk ([www.scienceguyrob.com](http://www.scienceguyrob.com/))

Institution: Edge Hill University

Version: 1.0

**Topic 7A.2 Part 2, Introduction Slide**

Hello and welcome to Part 2 of Topic 7A, Module 2, Advanced Python. In this module we aim to provide some background that will help you understand the programming you’ve already done. We eventually build upon this background and introduce some of the “advanced” features of Python. We’ll also explore the Object-Orientated approach to software design. My name is Dr. Robert Lyon, and I’ll be taking you through this module.

**Slide 1**

Part 2 will explain…

* The Python Software Landscape
* The different Python programming environments.
* Setting up your own development environment.

The aim: to get your ready to run the module activities, and setup your own development workflow.

**Slide 2**

* Learning Python can be complicated because there are so many things to keep track of:
  + The language – learning the syntax, keywords etc.
  + Coding in practice. How to write the code, where to write the code, how do things link together.
  + The interpreter. Where is it, how to use it?
  + Python libraries. What is a library, how to make use of it?
  + Code Editors. Which to use, and why?
* Let’s try to simplify the picture by breaking things down into three components First we have:
  + Language tools. These comprise the fundamental components of the Python programming language. This includes the interpreter, the error checking systems and built-in libraries.
  + External Libraries. These are modules of code written by individuals, teams, or companies, that don’t form part of the Python language as standard. You can install these to take advantage of code that is already written. Save us from having to write everything from scratch each time.
  + Tools include software that help with the development process. This includes error checking tools and code editors.

**Slide 3**

* To build and execute Python programs, we must have the language tools installed.
* To get them you can install them on your own personal computer.
* Alternatively, you can connect to a system that already has them installed, such as code academy.
* You can install different versions of Python – the latest is Python 3.7. It’s best to use the latest and greatest where possible.

**Slide 4**

* Code written by someone else to solve a problem/complete a specific task.
* External libraries and tools must also be installed when we need them.
* External libraries are often obtainable as “packages”.
* To make use of them, they need to be installed somewhere that the Python interpreter can find them.
* External libraries might include things like, code written by a company to help you access their files. For example, like a Microsoft Excel file reader package. This isn’t part of the Python language or built-in libraries, but it is useful, nonetheless.

**Slide 5**

* Writing code becomes tricky as programs become more and more sophisticated.
* Thus, we use purpose-written Python development tools to help write our code.
* We must install these too to use them.
* There are many such tools: PyCharm, Visual Studio, Eclipse, Jupyter... Personal preference usually dictates which tool people use.
* To help us manage our external python libraries, we can also use package managers.
* These managers simplify things for us, but they must also be installed.

**Slide 6**

* I Imagine this software ecosystem appears daunting at first – so many things to learn about! But we don’t have to let this scare us off.
* Environments like this can be provided for us, with all these software systems pre-configured.
* They packaged them up and host them from web-based computers.
* You can connect to these and run your code without issue. Just as you did with the Code Academy tutorials.

**Slide 7**

* Google Collaboratory is online environment that contains all the tools necessary to write and execute Python programs.
* To use the Collaboratory, all you’ll need is a google account, e.g. a Gmail Account.
* When you login to the Collaboratory, it creates a computer just for you to work with.
* The environment will allow you to create Python code, and execute it.
* The environment is special, as it allows code to be executed in an interactive fashion. This means you don’t need to write a whole source code file, before you can run some code.
* Instead you can execute individual commands one at a time. This is great for learning.
* Before proceeding, please create a google account if you don’t already have one.

**Slide 8**

* To open the colab environment, first just search for it via Google.
* Once you’ve found it, click on the link. This will cause the colab to start loading. It will take a few seconds, after all, its creating an entire Python environment for you to use.
* We can see the default page has some useful information and tutorials. I strongly suggest you read through these to understand the environment before proceeding. For those who are impatient, like me, I move swiftly on to opening my own colab “Notebook”.
* This creates an empty notebook with not much happening inside of it. You can see there are some option menus, and a single grey “cell” at the top of the page. The name of the file that has been created for this colab session is called “Unititled2.ipynb”. Don’t worry about what the file extension means for now.
* The cell at the top is a “code” cell. This means we can write code into it, and have it executed directly by the Python interpreter. Here I simply print out a string and execute it by clicking on the play button, to the left of the cell. This takes a few seconds to execute. Any code that is inside a code cell can be executed.
* You can create text cells in the colab too. You can write plain text into these, and such cells will not be executed as code. You can hop back into text cells, or any cells for that matter, by double clicking on them. I do this here to correct my mistake.
* Next I try to do some basic math to show you that the code is really being executed in the environment.
* Now I add another text cell. You can mix text and code cells any way you like. You can also add rich text formatting to text cells, as I show you here.

**Slide 9**

* You can delete cells in the notebook with ease.
* You can also move cells up, or down as you require.
* Notice how cells created at the bottom of the notebook, still know about variables created higher up. That’s because those variables are stored in memory, and the python interpreter knows they are there.
* However, if the notebook gets “interrupted”, or I restart the interpreter “runtime” directly myself, will I still see the variables? Let’s move the bottom cell to the top, now that we’ve restarted the runtime.
* If I now run this cell, I encounter an error. This is because this variable no longer exists in memory. The interpreter has been restarted, so it no longer knows about it.
* If I switch things around, and run the cell that declares the variable, I can correct things.
* This is important – variables or objects declared stay in memory for as long as the session is running. Variables declared at the top of the notebook will be accessible to those below. This means it is important to order cells correctly.

**Slide 10**

* Now suppose I create a variable “a” equal to 1.
* I then create a variable “b”, which is equal to “a” plus 1.
* Now I print out the value of “b”.
* If I then alter the value of “a”, this change impacts all cells – even those higher up. This is because the value is updated in memory.
* Thus, when I re-run the cell that defines “b”, I get a new value for “b”.
* The message here is to be careful with ordering of cells, and how you define variables. You may not want to overwrite the content of a variable, but you might by accident, if you don’t remember that the interpreter is storing these in memory for use anywhere in the notebook.

**Slide 11**

* In the Colab, the interpreter and built in libraries are hidden.
* There are no external libraries loaded by default – but you can load them for yourself – this is an advanced topic that we will cover later.
* The Colab doesn’t hide the tools at your disposal – it provides a friendly user interface via which you can interact with Python.

**Slide 12**

* The Colab environment is running a software tool known as Jupyter.
* Jupyter allows you to create interactive Python environments in the form of notebooks.
* Notebooks are great for teaching, and for running code interactively, just as I did in the videos.
* If I didn’t use Jupyter, I’d have to write source files directly, and pass them to the interpreter.
* Jupyter is a tool that I, and many scientists use day-to-day.

**Slide 13**

* You don’t have to install it – but it is easy to get.
* It has been packaged into a system called Anaconda.
* Anaconda contains everything you need to run Python programs. It includes,
  + The Python interpreter.
  + Built-in libraries.
  + A package manager that you can use to setup external libraries.
  + Jupyter for writing and running interactive Python code.
* Anaconda is easy to download and install.

**Slide 14**

* You don’t have to install it – but the video to the right will run you through installing your own version. That way you won’t need the Colab for other projects.
* Note: I strongly recommend anaconda, which is discussed in the video.
* The video also shows you how to use Anaconda and Jupyter.

**Slide 15**

* If you installed anaconda, there is another way you can write and run Python code.
* You don’t need to use all the tools Anaconda provides.
* You can download your own code editors and use those to create and run applications.
* This is how most software companies will create code.
* I’ll show an example of this next.

**Slide 16**

* Here we can see my computer screen. I have an Integrated Development Environment window open – this program is called PyCharm Community Edition. It helps me write Python code – it does syntax checking and the like.
* PyCharm lets me create coding projects, view files, and organise them efficiently.
* Here a create a folder to contain the source code for my project.
* Once the folder exists, I create a new source code file. I call it “Test.py”.
* I then write some code into the file – a simple print statement will suffice.
* Once I’ve finished writing my code, I need to pass the code file to the interpreter to run it.
* To do this, I move to a terminal window. The terminal is a simple interface that you can find on all Mac Computers. On Windows machines, it’s known as the command line (you can open it by typing cmd in the run box above the start menu on windows machines.).
* Once the terminal/command line is open, I move inside the src folder, and print the contents. I can see my Python file is there.
* Then I check that I have a Python interpreter. I check this by running the command: python --version.
* Here we see I have Python version 2.7 installed.
* I can now ask the interpreter to run the file via the command: python Test.py.
* We can see the output is as expected.
* This is a very simplistic example showing how to write and run code for yourself, assuming you have a Python environment installed. I include it so you can see how things can be done in practice. I don’t expect you to do this for yourself at this time.

**Slide 17**

* You can always install Python without Anaconda.
* This requires more effort and can be complicated. You have to download all the tools for yourself and manage the Python packages.
* I won’t show you how this is done as I don’t want to create confusion – it’s best to just install Anaconda.
* For those that are adventurous, here’s a video link that will walk you through the process:
  + For Windows: https://www.youtube.com/watch?v=ndrCfBJkkvE
  + For Mac: https://www.youtube.com/watch?v=TgA4ObrowRg

**Slide 18**

* Before we finish part 2, it’s worth reflecting on what we’ve learned.
* We now know that a Python development environment has 3 principle components:
  1. The Python interpreter and standard libraries. This is required.
  2. External libraries written by others that save us time. These are optional.
  3. Development tools such as package managers, and code editors. These too are optional but make our lives much easier.
* We know we can interact with Python environments on our personal PCs, or on machines hosted elsewhere which we connect to via the internet.
* Goolge colab is one such remote environment that happens to be useful for learning. We’ll use this in the remainder of the module.
* I appreciate some aspects may still be confusing – but stick with it. None of us pick this stuff up right away. It becomes easier as you become more familiar with the technologies.

**Slide 19**

Here we’ve introduced,

* The Python Software Landscape – comprised of the standard libraries, external modules, and development tools.
* Google Colab environment.
* Jupyter notebooks.
* Setting up your own development environment in a variety of ways.

Next we get back to the fun stuff – coding. For those that want a head start, here’s a link to the Colab notebook we’ll be working on next.

<https://colab.research.google.com/drive/1JNwsQ6PM7IifWK2fXMEa0PPjw2RjxIMj>

**Slide 20**

This slide contains some links for you to follow up in your own time. Once you’ve looked at those, move onto part three.

*Useful links*

* PyCharm community edition: https://www.jetbrains.com/pycharm/download/
* Eclipse IDE: https://www.eclipse.org/downloads/
* Visual Studio: https://visualstudio.microsoft.com/
* Anaconda: https://www.anaconda.com/
* Jupyter: https://jupyter.org/
* Another cool Python Course covering software installation to writing code: https://www.youtube.com/watch?v=\_uQrJ0TkZlc