Episode 1
Using the Interpreter

Anaconda

We recommend, but do not require, the Anaconda distribution from Continuum Analytics (www.continuum.io). An overview is available at https://docs.continuum.io/anaconda.

At the time the videos were produced, Anaconda distributed a frontend called Navigator. Through Navigator you can launch applications, manage your installation, and install new applications. Continuum provides documentation for Navigator at https://docs.continuum.io/anaconda/navigator.

For an easy-to-use tool you can use from the command line, try conda. From Windows, run conda in the command prompt. From Mac OSX or Linux, open a terminal (Apps/Utilities on OSX). The conda cheat sheet (https://conda.io/docs/_downloads/conda-cheatsheet.pdf) is particularly helpful. Using conda you can update everything from a single package to the entire Anaconda distribution. You can also set up your environment to run both Python 2.7 and a version of Python 3.

It is important to install Anaconda "Install for Me Only." You can find detailed installation instructions for your platform at https://docs.continuum.io/anaconda/install.
It would be a good idea to open a command line and type
conda install anaconda-clean
so that you can completely uninstall the distribution should the need arise.

Spyder

We do strongly recommend the free, open-source Spyder Integrated Development Environment (IDE) for scientific and engineering programming, due to its integrated editor, interpreter console, and debugging tools. Spyder is included in Anaconda and other distributions, or it can be obtained from Github (https://github.com/spyder-ide/spyder).

The official Spyder documentation is at https://pythonhosted.org/spyder; however, we recommend that beginners start with the tutorial included with Spyder. From the Help menu select Tutorial. The tutorial will open in the Object Inspector. You can follow along by scrolling and expanding the sections. Topics covered in this episode are described in more detail in the sections *Shortcuts for useful functions* and *Plotting*. This tutorial was prepared by Hans Fangohr of the University of Southampton in England.

After you have gained some experience, you may wish to try other IDEs. An Internet search should show several possibilities, though not all of them are completely free.

Troubleshooting

Several things can go wrong with your installation. Navigator may crash when you attempt to open it, especially on Macs. On a Mac you can start Spyder from the Terminal application by opening a window and typing

```
spyder&
```

This will enable you to continue to follow along while you try to resolve the Navigator problem. First try updating using conda

```
conda update conda
conda update anaconda
```

If that doesn't work, uninstall with
```
anaconda-clean –yes
```

Then re-download and reinstall.

More troubleshooting tips are at
https://docs.continuum.io/anaconda/troubleshooting
You can also join the Anaconda Google group and request help. Also check whether your problem has been reported at
https://github.com/ContinuumIO/anaconda-issues/issues
If you confirm you have a new issue, you can enter a bug report at the above site.

Episode Summary

We explored several features of the Spyder development environment. The default appearance of Spyder has three panes: the Editor, where you can type scripts; the Explorer, which has three tabs for Object, Variable, and File Explorers; and the iPython console. We used the menus to create a new file (File->new file) and to save it (File->Save as).   We explored printing from the script in the editor pane of Spyder, and printing expressions directly in IPython. If you type
```
print "Hello World"
```
In the Editor pane, you must run it for the print to be carried out.

You can also type commands directly into the iPython console. If you type an expression its value will be printed:
```
In  [1]: x=5
In  [2]: y=7
In  [3]:x+y
Out [3]: 12
```

This is not the case for expressions typed alone on a line in the Editor pane.

```
x=5
y-7
x+y
```

Nothing will happen if we run the above lines of code.

In the iPython console we can also use up-down arrow keys to scroll in our commands, and the right-left arrow keys to edit them.

We learned to run scripts either with the green arrow icons or through the Run menu. *Run/green arrow* runs the entire script. *Run selection or current line* will run a highlighted portion of our script. We can create cells by enclosing chunks of code with lines consisting of

```
#%%
```

Run cell/green arrow with a box runs the cell. *Run cell/green arrow, box, and red "line feed" arrow* runs the cell and advances.

A yellow triangle beside a line indicates a syntax error or potential problem. Spyder offers tab completion for names familiar to it. It can show a list of members of a package for your selection, and when you have chosen a function it can show you a list of its arguments.

One of the handiest features of Spyder is the ability to indent and unindent blocks of code from the Edit menu. Click to highlight the text you wish to indent or unindent, then choose the appropriate option under Edit. We will later see why this is very useful. Another helpful option allows you to highlight several lines of code, then select "Add block comment." This will save you a lot of typing once you begin to write and debug your own programs. To remove the block comment, select it and choose "Remove block comment" from the Edit menu.

We can see some of these these features in action by creating a simple plot Type

```
matplotlib.pylab as plt
```

First we see a yellow triangle, indicating plt is imported but not used. We ignore it since we know we will be using it. As we type

```
x=plt.
```

We see the editor show us our choices from the `pylab` package. we can select one or we can keep typing. We type

```
x=plt.linsp
```

to further narrow it down. That leads us to

```
x=plt.linspace
```

The editor then pops up a box with the arguments required by `linspace`. Finally we type inside the parentheses

```
-1.*plt.pi,plt.pi, 100
```

for a final result of

```
x=plt.linspace(-1.*plt.pi,plt.pi.,100)
```

Finally we type

```
y=plt.sin(x)
plt.plot(x,y)
```

When we run this code, we see the plot appear embedded in the iPython console. We can right-click on the image to bring up a menu that allows us to save the plot.

The Variable Explorer allows us to show the values of variables in our programs. It is particularly helpful for looking at a group of values (an array). We can change the number of decimal places printed by clicking "Format" and typing in an expression of the form `%.3f` for three decimal places. The Variable Explorer also includes icons for saving, refreshing, or importing data to our workspace.

To clear all values in the workspace, type

```
%reset
```

at the iPython console.

Now re-run your sine-plotting code and observe how the variables acquire values. We can right-click on the `x` values to see the full array. We can use the Format menu as described above to change the number of decimal places displayed for a floating-point number. If we ask for more decimal places than can be computed in the computer's hardware, we will find that the “extra” spaces will be filled with zeros. We can see this if we click on the variable representing `pi` and request `%.64f`. We know that pi has an infinite number of decimal places, but the approximation we see is limited to some finite number and the rest are represented by zeros. This clearly shows us the limitations of computing with finite representations of the mathematical real numbers. We can see another consequence of this by looking at the first 64 digits of the first entry in our `y` array; it should be exactly zero, but is not.

We can save data through the Variable Explorer. The right sidebar of the Variable Explorer shows several icons. Hovering over each one shows what it does; we can refresh, refresh periodically, save, save as, import, and select options. As an example we can use the save as icon to save the variable `y` as `sineData.pydata`; we can then clear our workspace with `%reset` and use the Variable Explorer import icon to reload it.

From the File Explorer we can browse through icons on the menu bar to change to a different folder for our programs. Spyder's top text bar shows the *working directory* (or working folder) for our programs and other files. The File Explorer displays the contents of that folder. You can double-click on any file in the File Explorer to open it. If it ends in `.py` it will open in a new tab in the Editor pane. Otherwise you will be asked how you want to handle the file.

We've now learned the basics of working with our chosen Integrated Development Environment. Several others are available for Python, some of which are free and others of which require payment at least for certain features. Popular alternatives to Spyder include PyCharm (“freemium” i.e. you pay for certain features. www.jetbrains.com), Rodeo, which is similar to Spyder (http://blog.yhat.com/posts/introducing-rodeo.html), and the PyDev plug-in for Eclipse (www.pydev.org).

Further Resources

Spyder contains its own built-in tutorial. It tends to expect some proficiency with Python and discusses concepts we haven’t seen yet; or it may expect users to be beginners at Python but not at programming; however, it is worthwhile to go through at least part of the tutorial. To access it, start from the Help menu and select Spyder tutorial.