

A Short History of Visualization

...

Visualization and Computers

...

Elements of Programming

...

The PYTHON Programming Language and Environment

...

Let's Get Visualizing!

These first few activities show you how to set up the software and help you get familiarized with the software we will be using to do visualization. This will involve three steps:

1. Setup
2. Explore: Our First Visualization
3. Getting Fancy

Setting up Python and Pylab

A little further down the line we will be learning a bit of the nitty-gritty that goes into computer-based visualization: the basics of generating graphics and other figures through programming. For the most part, however, we are going to take advantage of the fact that software already exists that will allow us to manipulate and visualize data without worrying about the details involved in generating basic graphics or doing. The software we will be using is called **Pylab**, which is a 2-D plotting library that allows us to interactively manipulate a variety of visualizations in conjunction with the Python programming language. Instructions for installing Python and Pylab on your personal computer can be downloaded from the course website. The software should be up and running on all lab computers. For now, because we will be assuming that you are using the department computer lab, all instructions are for a PC running the Windows operating system.

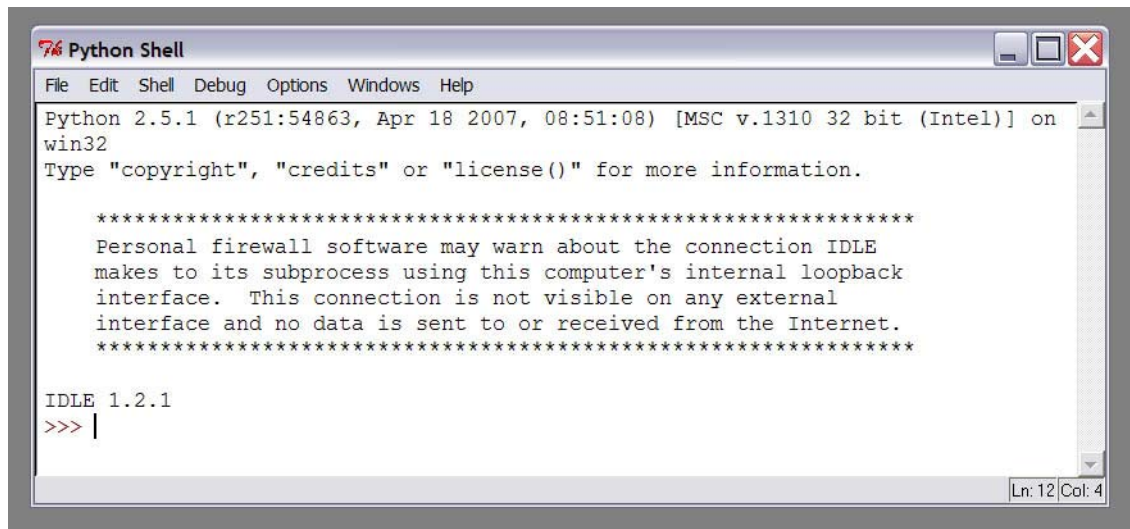
When the software was installed it should have created a short-cut to the Python **I**ntegrated **D**eve**L**opment **E**nvironment, or **IDLE**. The location of the short-cut will depend on your operating system (OS). On the Windows platform the short-cut can be found on the Start Menu, in the Python 2.5 folder. We will be using the IDLE environment to edit, compile, debug, and execute all of our Python code. To use IDLE first create a folder on the storage device (personal hard-drive or flash drive if in the lab) where you plan to store all of your visualization programs. Then, copy the IDLE short-cut (IDLE (Python GUI)) to the file folder.

Before we begin exploring the IDLE environment, we need to edit the settings of the short-cut so that Python knows where to find the programs that you will write. Navigate to your Python folder and select the short-cut that you just pasted (single-click). Open the Properties (in Windows: right-click and select `Properties` from submenu) dialog. There should be a field labeled `Start In`. Delete `C:\Python25\` and click the OK button (i.e., the field should be empty). Now Python will know to look in the current directory (your folder) for any programs that you write.

Exploring the Python Environment

Now that we are done installing and configuring the software, it is time to start taking a look around. In this exercise we will get oriented to Python through a simple visualization.

1. Start Python: If you haven't yet, download the [StartPython.py](#) file from the website and save to your Python directory. Right-click on it to bring up the submenu and select `Edit in IDLE`. If you wish to use Pylab, you must open IDLE in this way. Python is a relatively new language, and interfaces such as Pylab are still under development. Therefore, occasionally we will run into bugs, or instances where IDLE does not work as expected. If you run IDLE by just double-clicking on the short-cut the program will crash when you display visualizations. Two windows will appear on the computer screen; you can close the one titled `'StartPython.py'`. This will leave you with the following, which is the interactive Python programming environment, IDLE:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2.1
>>> |
Ln: 12 Col: 4
```

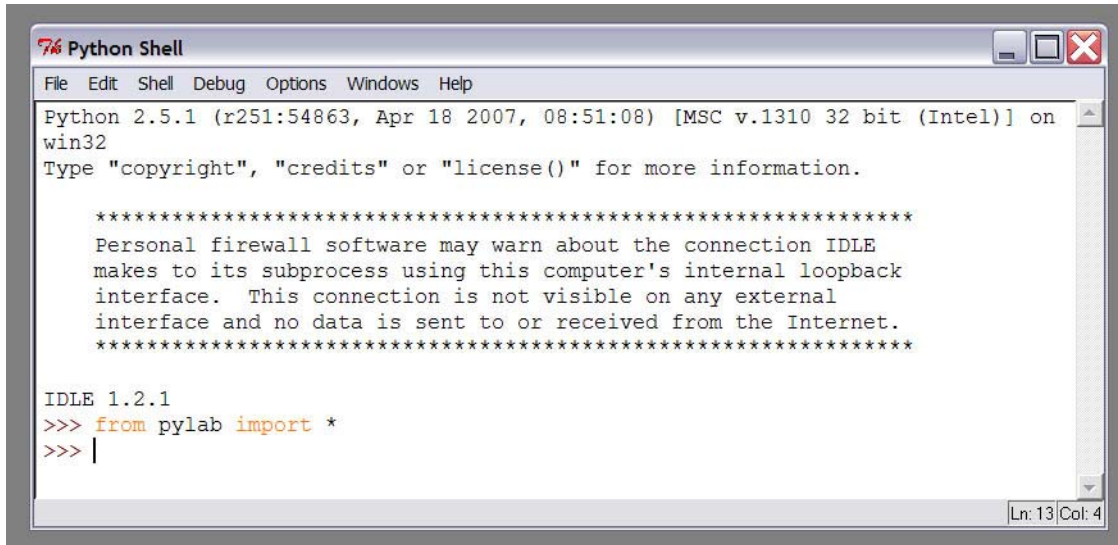
What you see above is the Python interaction window or the *Python Shell*. This particular shell is called *IDLE* (notice that it reports above that you are using IDLE Version 1.2.1). You will be entering all Python commands in this IDLE window. The next step is to load the **Pylab** interface and libraries.

2. Start Pylab: To start visualizing information in Python you need to first load Pylab, which provides graphical interface and set of related commands that allow the

development environment to 1) perform complex data manipulations and 2) produce visualizations. To start the visualization software, enter the following command in the Python Shell (where `>>>` is the prompt in the shell and does not need to be typed):

```
>>> from pylab import *
```

This interaction is shown below:

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following content: "Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", a large block of asterisks, a warning message about a firewall connection to IDLE, another block of asterisks, "IDLE 1.2.1", and the command prompt sequence: ">>> from pylab import *", ">>> |". The status bar at the bottom right shows "Ln: 13 | Col: 4".

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2.1
>>> from pylab import *
>>> |
Ln: 13 | Col: 4
```

This command informs the shell that you will be using the Pylab library. This import statement/command is something you will use each time you want to do visualization. After issuing the import, the shell is ready for the next command.

3. Get Data: To visualize information, we first need some information. From the course website, download the file **DrugData.py** and save to your Python directory. This file contains a simple dataset that tracks the proportion of twelfth graders from 1979 to 2007 that at some point used various narcotic substances. Load the data into the Python Shell by entering the following command:

```
>>> from DrugData import *
```

As when we loaded the Pylab interface, this statement tells the shell to import all information stored in the file, in this case `DrugData.py`. Okay! Now it is finally time to generate our first visualization.

3. Create a Visualization: There are many types of visualizations, but for now let's just do a simple one that illustrates the relationship between two variables. To do this we use the following command in the Python Shell:

```
>>> plot(x, y)
```

where x and y are the two sets of data you want to compare. The example below shows how to issue this command for two sets of data from the `DrugData.py` file: **Year** and **AnyDrug** (proportion of students that used any type of narcotic substance):

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

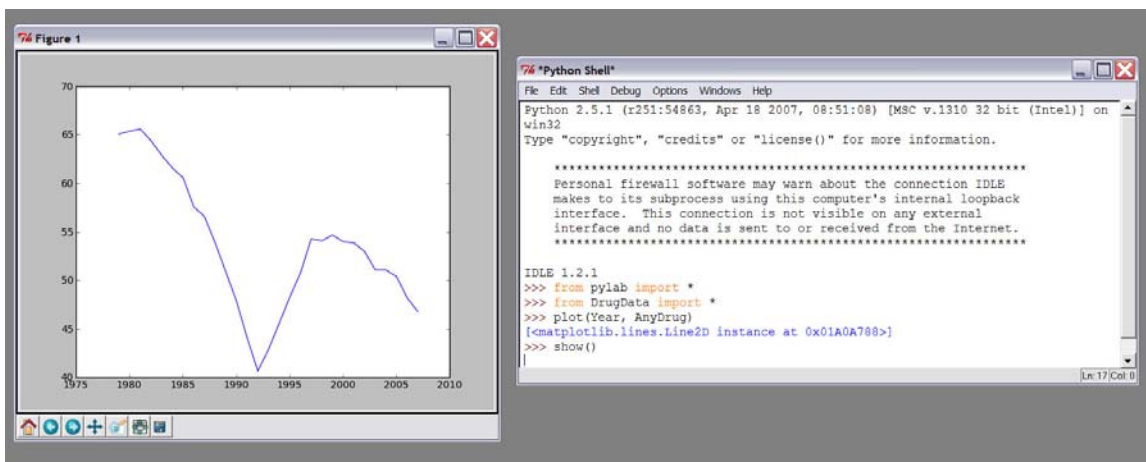
*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2.1
>>> from pylab import *
>>> from DrugData import *
>>> plot(Year, AnyDrug)
[<matplotlib.lines.Line2D instance at 0x01A0A788>]
>>>
```

The Python Shell reacts to the command by informing you that a visualization (in this case a linear, two-dimensional visualization) has been created and stored at a specific memory location (in this case `0x01A0A788`). Notice, however, that no graphics have actually been drawn on your screen. That is because you have not yet instructed the computer to display the graphic. To display your plot, enter the following command;

```
>>> show()
```

Executing this command will open a new window, or **figure** in which your graphic is now displayed:



Pylab figures are completely interactive and can even be saved as various graphic types. Click on the buttons at the bottom of the figure window to explore what each does. When you are done, close the window by clicking on the close window button (X) in the top-right corner or by executing the close command:

```
<<< close()
```

Congratulations! You have just created your first visualization and in doing so, written your first Python program! To review what we did, each visualization session will begin by starting the Python software by choosing to edit the StartPython.py file in IDLE, followed by importing the Pylab software. Next, you will usually load some data into the Python Shell. From then on, you can issue any commands needed to manipulate and visualize your data.

The Pylab interface contains many commands that enable a variety of two- and three-dimensional visualizations. We will be learning many of these as the semester progresses. The important thing to remember is that all of these commands are issued in the Python programming language. Thus as you learn to visualize, you will also be learning to program in Python.

Programming languages have a very strict way of typing commands. As you may have already experienced, the language is very precise: every parenthesis, quotation mark, and the case of each letter that makes up a command has to be type exactly as described. This set of rules defines the syntax of the language, while strict, it is fairly simple and easy to get used to. This precision is required so that there is exactly one interpretation of each command and the computer can exactly determine the action that should result. Thus, programming languages differ from human languages in that they are *formal* (as opposed to *natural*).

Getting Fancy

Now that we have made our first visualization, let's try getting a little fancy. Pylab has facilities that let you interactively customize many aspects of every graphic it can produce. In this exercise, we explore some of these customizations.

Line Properties: The simplest way to add flair to a simple visualization is to manipulate both the color and style of the plot lines. In Pylab we manipulate many types of line properties, including: the line width, dash style and color. Lines can also be labeled and individual data points can be emphasized by various symbols (or markers), whose properties (color, size, and shape) can also be independently manipulated.

Line Property	Values
alpha	alpha transparency on 0-1 scale
color	a Pylab argument
label	a string to be used for a legend
linestyle	one of -, --, .., -.
linewidth	a float, line width in points

marker	one of +, o, ., s, v, x, >, <, ^
markeredgewidth	line width around marker
markeredgecolor	color of marker edge
markerfacecolor	color of marker face
markersize	a float, marker size in points

There are several ways to set line properties. For now, we are just going to set them using the plot command that we used earlier and keyword arguments.

Let's start by changing the line style from a solid line to a dashed line by executing the following command:

```
>>> plot(Year, AnyDrug, '--')
```

Alert: if you are continuing your prior session, this will automatically display a plot. Otherwise, if this is a new session you will need to follow the plot command with a show command.

Perhaps you do not care for the dashed line and want the style to be dotted instead. To re-visualize your plot, first you need to either close (`close()` command) or clear (`clf()` command) the figure. Clearing the figure allows you to use the same graphics window to visualize another graphic. If you close the figure, new graphics will be visualized in new windows. To clear the figure and then replot with a dotted line, execute the following:

```
>>> clf()
>>> plot(Year, AnyDrug, '.')
```

Congratulations! Between the prior exercise and this one, you've now just visualized the same data in three different ways. Let's add a little more flair by changing the color of the line. Pylab can plot basic graphics (lines, markers, and texts) in several colors, each represented by a single letter (b, g, r, c, m, y, k, w).

As with line styles, we can change the line color directly using the plot command. For example, to make our dotted plot have red, instead of blue points, clearing the prior visualization and then execute the following:

```
>>> plot(Year, AnyDrug, '.r')
```

b: blue
g: green
r: red
c: cyan
m: magenta
y: yellow
k: black
w: white

Or, we can change the marker from a dot to another symbol by replacing the . with one of the symbols listed in the line property table. For example:

```
>>> plot(Year, AnyDrug, 'sr')
```

will generate a scatter plot with red squares indicating each data point.

Line styles and markers can be combined. The following generates a scatter plot whose points are connected:

```
>>> plot(Year, AnyDrug, '-rs')
```


To modify some of the other line properties, or to get a little more complicated (say red squares with a black line) a little more context needs to be provided through the use of **keyword arguments**. For example:

```
>>> plot(Year, AnyDrug, color='k', marker = 's', markerfacecolor = 'r');
```

plots a combination scatter and line plot with red squares and a black line. Or

```
>>> plot(Year, AnyDrug, linewidth = 4.0)
```

reproduces our original plot but with a much thicker line. It is easy to see how through some very simple manipulations we can create some very fancy graphics!

Explore a little further

Okay! Now you are on your own! Computer programming is in many ways a skill and the only way to get better is to Practice! Practice! Practice! Spend some time experimenting with the various features introduced in this chapter. Experiment with all the different types of colors, markers, and line styles. Make scatter and line and combination plots of the data. And, while you are at it, spend some time thinking about how these very simple manipulations change the way the data are presented.

PYTHON Review

...

IDLE Hints

...

Glossary

...

Exercises

1. Explore the Real Time Rome Visualization (<http://senseable.mit.edu/realtimerome/>). Read the project description on the website and then browse to the snapshots and movies (under the SCREENS heading). Choose one of the six visualization examples to examine a little more closely. What type of visualization is this (Communication, Discovery, Insight, Aesthetic). Write a short essay (1-2 paragraphs) explaining the visualization, its purpose, and whether or not you think it is a “good” visualization. Consider: what makes a visualization “good” or “bad”?

In your Python folder, right click on your DrugData.py file and select **Edit with IDLE** to open the file in IDLE. As you can see, this file contains more data than was explored in the example: in addition to AnyDrug, the file also contains data on the following substances: Marijuana, Hallucinogens, Cocaine, Heroin, Cigarettes, and Alcohol. Close the file (without saving) and return to the Python Shell. Initialize the session (load Pylab and the DrugData).

2. Plot the use of Marijuana versus time using a simple line plot. It would be interesting to compare the use of Marijuana through time to that reported for AnyDrug. To illustrate multiple plots on a single figure, we can use the command `hold()`.
Executing:

```
>>> hold(1)
```

tells Pylab to plot the next figure on the same set of axes that is already illustrated. To release the figure just execute the command `hold(0)`.

Hold the figure and then plot AnyDrug versus time using a different color than the line for the Marijuana data. What does this comparison tell you about the use of Marijuana? Save the figure.

3. Repeat #2 using one of the other datasets. What does this comparison tell you about the use of that narcotic through time? Save the figure.
4. Fancy it up a bit. Choose one of the narcotics and plot versus time. Modify at least three line properties to spruce up the image a bit. Save the figure. How does this manipulation affect the impact of and information conveyed by the visualization?
5. Pylab provides many other types of relational visualizations in addition to the simple scatter and line graphs generated by the `plot()` command. These include:

```
>>> bar(x,y)
>>> barh(x,y)
>>> stem(x,y)
```

as well as means for visualizing a single variable:

```
>>> hist(y)
>>> boxplot(y)
```

Choose two of the drug variables and experiment with these types of plots. What type of graph does each produce? For the relational visualizations compare the drug usage data to time (Year) and for the single variable visualizations just use the drug usage data. How does the choice of visualization affect the message conveyed? Does it ever alter the type of the visualization (i.e., from Discovery to Communication or Insight)? Save your favorite graphics.