CE 549 Python Lab 1 - Introduction to Python Programming

Prepared by
Zhu Liu and Venkatesh Merwade
Lyles School of Civil Engineering, Purdue University
vmerwade@purdue.edu

March 2018

Objective

The objective of this session is to get exposure to Python programming within PyScripter environment, and write some simple code to access data and plot it using some key Python libraries.

Learning outcomes

1) Write simple Python code inside PyScripter to create random numbers and frequency histogram using the numpy library
2) Create graphs in Python using the matplotlib library
3) Access hydrology data from CSV files using the Pandas library, and create stage-discharge rating curve and streamflow hydrographs using the matplotlib library.

Data: The data you will need for this module is available on blackboard for Purdue students. Others can get the data from this link: ftp://ftp.ecn.purdue.edu/vmerwade/download/data/wabash.csv

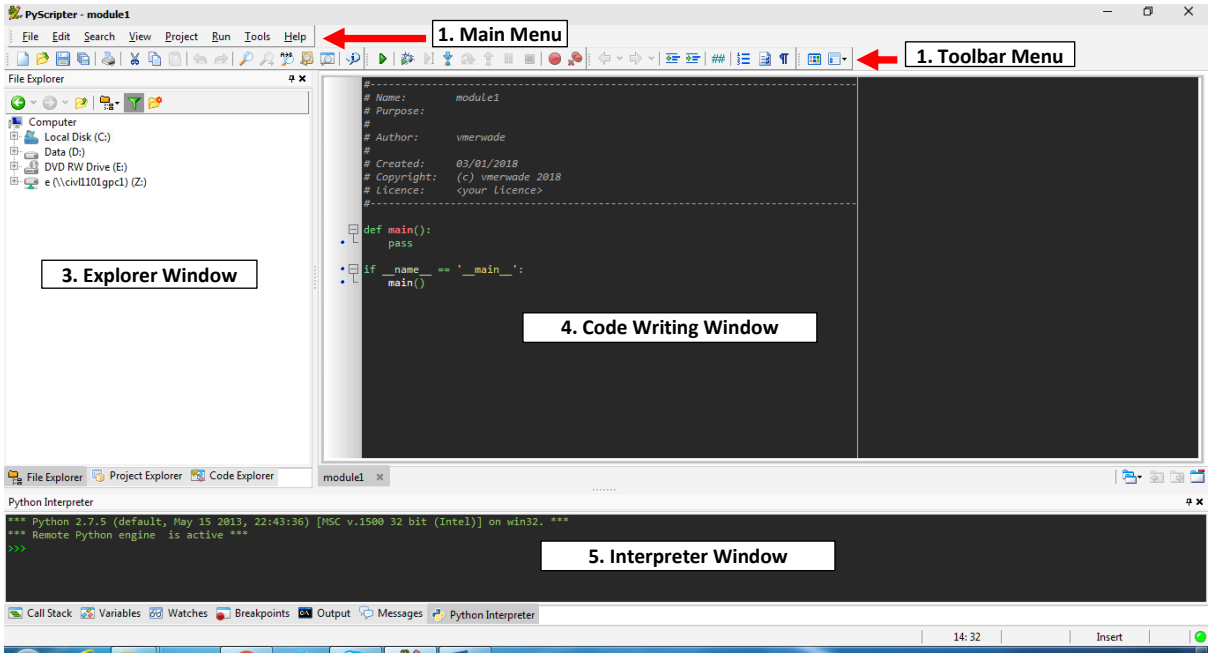
Introduction

When it comes to programming, adopting a language depends on what you want to accomplish. For example, if you want to write a code to solve complex numerical equations, you may use C++ or Fortran. Similarly, if you are interested in statistics, R may be a good choice. Accordingly, when it comes to automation of GIS analyses, python is widely used as it is a higher level language that is open source, cross-platform, and is easy to learn and code. Additionally, standard GIS programs such as ArcGIS and QGIS have tools and functions that can be automated using Python. You can find more about python at http://www.python.org/doc/ and http://www.diveintopython.org.

Now, you need an "environment" to write your Python code. If you are an ArcGIS user, there is a Python scripting window in ArcMap that can be used to write your code for automating geo-processing tasks. We will learn more about that later, but there are many other independent softwares, such as IDLE, Spider, Sublime, and PyScripter, to write and run a Python code. In this class, we are going to use PyScripter just because of in-house experience and expertise within my research group.

Getting PyScripter (Purdue students can skip the download step and just open the program from the Start menu). **Download** PyScripter from https://sourceforge.net/projects/pyscripter and install it on your computer.

Start/open PyScripter by double clicking the icon or from the start menu. The PyScripter window, shown on the next page, allows you to write the Python code, run the program, and look at the results. The PyScripter window has four sections as shown in the Figure below. (Note: your background color in PyScripter can be white or black; it does not matter)



Pyscripter has five main sections:

(1) Main Menu allows you to create a new file, print, zoom in or out and search for help.

(2) Toolbar Menu has several Common tools to run the code, search, cut/paste, etc.

(3) The Explorer allows you to explore files similar to windows explorer, explore your current python project, or explore your code to see the name of the libraries, subroutines, etc.

(4) Code writing window is where you will write your code to create your .py file, modify it, save it and run it.

(5) Python interpreter is where you will see the results (more often errors!) after running the code.

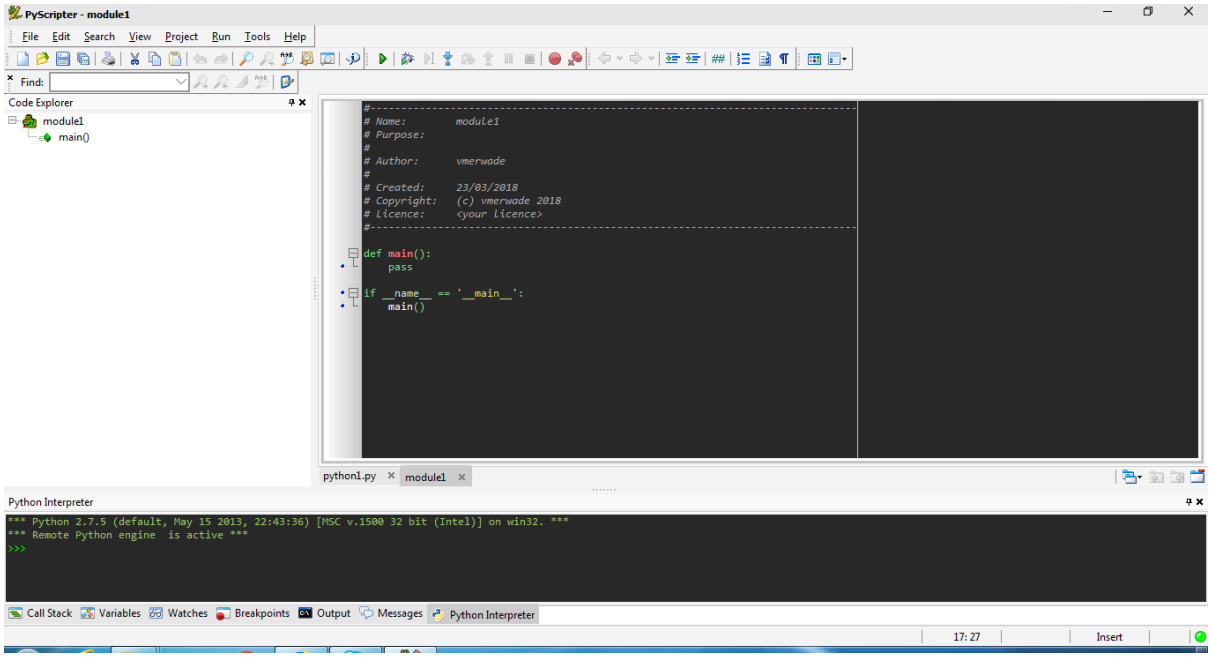
Python Libraries

A library is basically a collection of standard code or sub-routines that are frequently used in programming. A library helps a coder (you) to accomplish a programming task in a few lines of code by borrowing all the lines stored in a function within a library. One of the greatest strengths of Python is its large library that provide access to tools for numerical computations as well as GIS applications. Some common libraries that we will be using in this class are briefly introduced below, along with a sample code that you can try in your PyScripter.

(i) numpy

If you are familiar with Matlab, numpy provides Matlab like functionality within Python. Using numpy you can solve multi-dimensional arrays and matrices, and perform simple to complex mathematical operations including statistics. Now, lets see how to use numpy in PyScripter to generate 1000 random numbers from a normal distribution with mean = 100 and standard deviation = 15.

When you open PyScripter, you will have an empty module as shown below.



If you do not see an empty module, you can create a new file by using the Main menu. Click on File-→New-→New Python Module. Save this module as "python1" inside a folder called M1.

The initial lines that start with # are just comments, and are not considered as code. You can write any information about your script by using # before any text/comment. This is our first code so for now, we will just leave the default text unchanged. Delete everything after the last line with #. Specifically, the line that starts with def main(), and everything after that.

Write the first line of code as below. (Note: All code in this tutorial will be highlighted to distinguish it from the regular text)

```
import numpy as np
```

Using this single statement, we have imported or borrowed the *numpy* library and referenced it as np. This is how we will import any library. You can use any name to reference a library. In this case we used np as it is an abbreviation of numpy. You can use nmy, npy or anything. You get the point! Use something logical!

Now we are going to define two variable for the mean and standard deviation, and assign them the value of 100 and 15, respectively. Again, you can pick any name for defining a variable, but it is a good practice to use something that is self-explanatory. We use greek letters □ and □ for mean and standard deviation, respectively. Lets write our second line of code to define these two variables as below.

```
mu, sigma = 100, 15
```

Here we defined both variables in a single statment. We could also do mu = 100 on one line and sigma = 15 on another line. Once we have the mean and standard deviation, we can then generate values using the following statement.

```
x = mu + sigma*np.random.randn(1000)
```

The randn function above generates random numbers from a standard normal distribution (mean = 0 and variance = 1), which is then scaled to our mu and sigma, and stored in variable *x*. Basically, *x* is an array to store 1000 random values from the specified normal distribution.

Q. What is an array?

Now run the code by clicking on the green triangle button in the toolbar (ctrl + F9). In the lower left corner, you will see “Running” and then "Ready" or “Script run OK”. This means the code ran successfully, and you can now see the results.

We defined an array named x to store our random values. You can see the results by calling this x array in Python Interpreter window. Simply type x in the Pyton interpreter window, and press *Enter*. You should then see all the random values as shown below.

```
>>> x
array([  99.43481397,  119.01639987,   91.37194674, ...,   94.69074833,
         86.805851  ,   92.4282399 ])
>>>
```

Congratulations! You just finished your first code in Python! Lets experiment with another library in Python.

(ii) Matplotlib

matplotlib is a plotting library for the Python programming language and it can be used to make plots in PyScripter. Lets use this library to create a histogram from the x array we just created. Import the pyplot sub-library from matplotlib and name it as plt. Write the following code in your pyscripter window as shown below. (Note: you can either write this statement after your last statement, but it is a good practice to have all imports on the top so you know what libraries are available in the code.)

```
import matplotlib.pyplot as plt
```

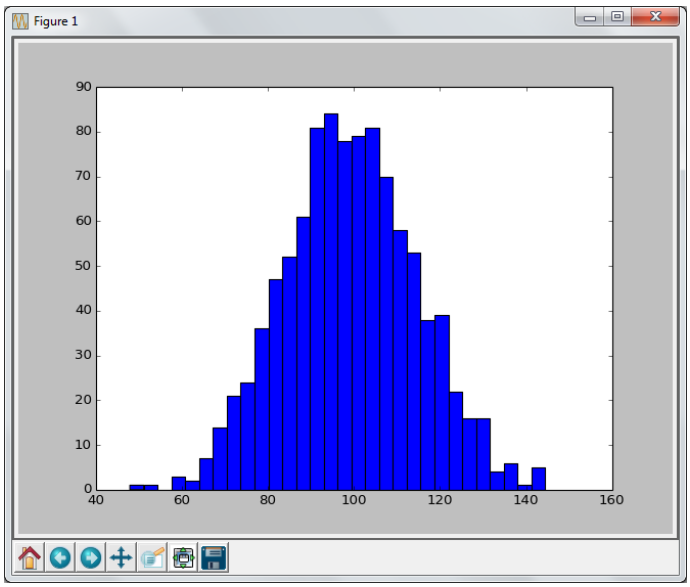
Now lets create the histogram of x by using plt. Write the following code in your pyscripter window. This will generate the histogram by dividing the array into 30 bins. Bins size is a parameter so you can choose some other value if you wish.

```
plt.hist(x,bins=30)
```

After creating the histogram, lets plot the histogram by using the following code.

```
plt.show()
```

Now run the code, and you should see something similar to the following histogram.



If you want, you can play around with the bin size to see how the plot changes.

Lets use pyplot to create scatter plot between two random variables. Define two random variables *a* and *b* as shown below. Remember we used *randn* earlier to create random numbers from a standard normal distribution. The random function will generate purely random numbers. The number inside the parenthesis defines the size.

```
a = np.random.random(100)
```

```
b = np.random.random(100)
```

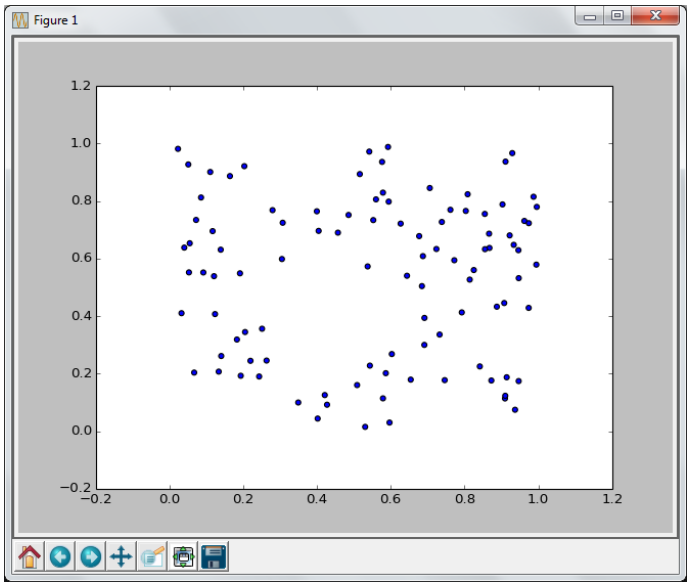
Now plot a and b on a scatter plot by using the code below. After you type plt, you will see a list of associated functions so pick scatter and provide a and b as the two variables as shown below.

```
plt.scatter(a,b)
```

Then show the plot.

```
plt.show()
```

You are ready to run the code at this point. After you run the code, you will first see the histogram, which you have to close, and then the scatter plot will be displayed as shown below.



How can you avoid the histogram to show and let the program only show the scatter plot? Think of #

(iii) Pandas

Python Data Analysis Library, or pandas, is a Python package providing fast, flexible, and expressive data structures designed to make working with is “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. (Note: Ideally you should have Pandas installed, but if you think it is not installed, you will have to install the pandas library to proceed. Instructions for its installation are beyond the scope of this module. You can find that information on the web). Lets use Pandas to create a stage discharge rating curve for the Wabash River gauging station in Lafayette, IN. one year of stage discharge data for this gauging station is provided as a CSV file in your Python1 folder on Blackboard. Download that file and save it in your working directory. This module will use c:\temp as its working directory so make sure you use the appropriate directory name in your code.

As usual, lets first import the *pandas* library as *pd* <all import statements will be at the beginning of your code>

```
import pandas as pd
```

Now read the csv file from your working directory and store it in data as shown below (note the forward slash in the directory name). *usecols* will store the data from each column in the respective array.

```
data = pd.read_csv('C:/temp/wabash.csv', usecols=['datetime', 'discharge', 'stage'])
```

The above statement reads the CSV file and stores the data in three arrays: time stamp in 'datetime', flow values in 'discharge', and stage in 'stage'. Because the datetime has both date and time in hh:ss format, it is stored as a string (or text) instead of date. Lets convert this array from string to date by using the statement below. This statement will convert the string to a datetime format array and stored it in timestamp series.

```
timestamp = pd.to_datetime(data.datetime)
```

Now we can plot information from data by using the statement below.

```
plt.plot(data.stage,data.discharge)
```

The scatter function we used earlier gave us points. The plot function will give us a line. Lets also define our x and y axes labels, and provide a title for the plot as shown below. The functions we are using here are self explanatory.

```
plt.xlabel("Stage(ft)")
```

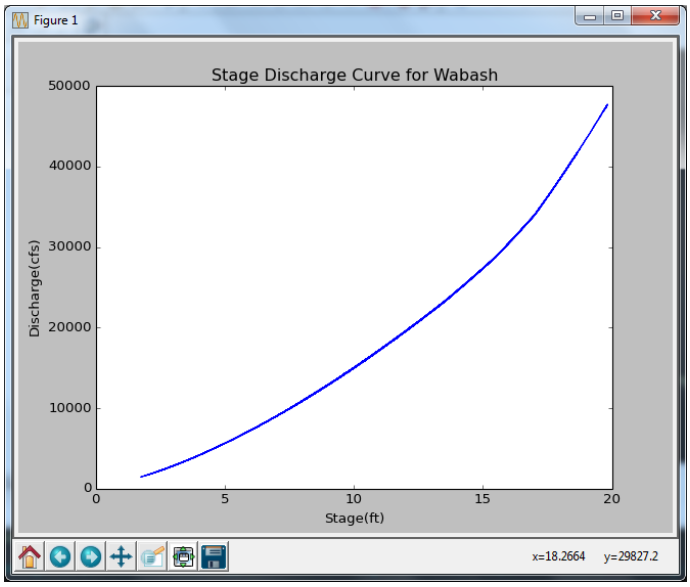
```
plt.ylabel("Discharge(cfs)")
```

```
plt.title("Stage Discharge Curve for Wabash")
```

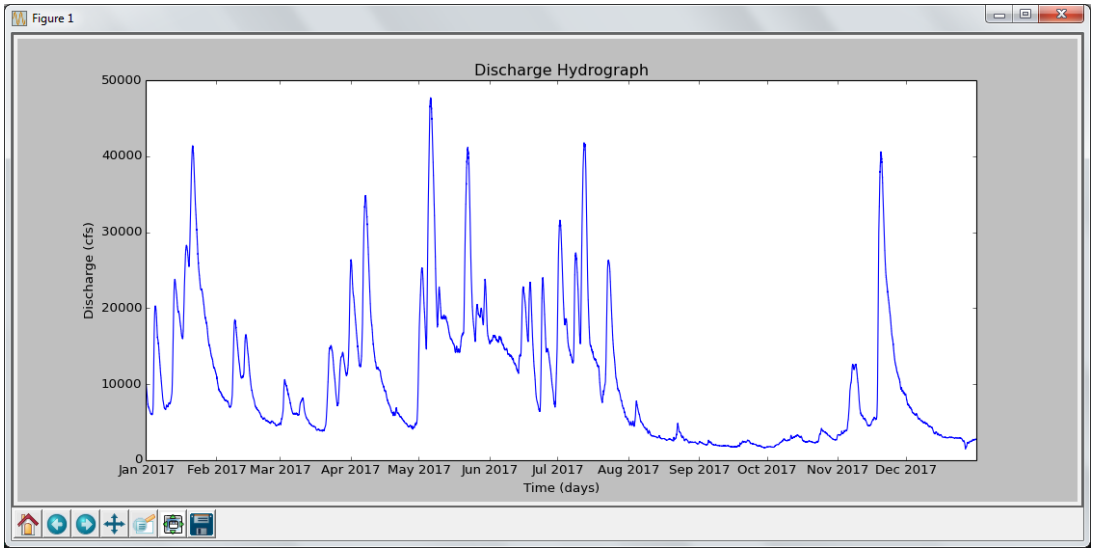
Finally, show the plot by using the statement below.

```
plt.show()
```

Run your code. If you want you can tell the program to skip the earlier plots by commenting the related statements using #. The stage discharge curve is shown below.



Now write some python code on your own to create a discharge time series as shown below. Make sure it has axes and plot titles.



You are doing wonderful! One more library that we will be using a lot is the ArcPy, which is briefly introduced below, but will be used later.

(iv) ArcPy

ArcPy (often referred to as the ArcPy site package) provides Python access for all ArcGIS geoprocessing tools, including extensions, as well as a wide variety of useful functions and classes for working with and interrogating GIS data. Using Python and ArcPy, you can develop an infinite number of useful programs that operate on geographic data. ArcPy is supported by a series of modules, including a data access module (arcpy.da), mapping module (arcpy.mapping), an ArcGIS Spatial Analyst extension module (arcpy.sa), and an ArcGIS Network Analyst extension module (arcpy.na). ArcPy classes, such as the Spatial Reference and Extent classes, can be used as shortcuts to specify geoprocessing tool parameters that would otherwise have a more complicated string equivalent. In ArcPy, all geoprocessing tools are provided as functions, but not all functions are geoprocessing tools. In addition to geoprocessing tools, ArcPy provides a number of functions to support geoprocessing Python workflows. Functions (often referred to as methods) can be used to list certain datasets, retrieve a dataset's properties, validate a table name before adding it to a geodatabase, or perform many other useful scripting tasks. We will learn about these in future modules.

Homework

Create a python module called hm1 (home module 1) to create a histrogram and discharge time series of the daily streamflow data for your project site.

Turn-in (Due date 03/28/2018)

1. Page 1: Your python code (include comments to explain your code)
2. Page 2 : a neat histogram with axes titles and hydrograph with axes titles