NORTHERN ILLINOIS UNIVERSITY

PHYSICS DEPARTMENT

Physics 374 – Junior Physics Lab Spring 2024

Python Tutorial #2

**Variables & Arithmetic**

In this lab, we will perform some basic math functions to get an understanding of variables.

In programming languages like Basic, C++, and Fortran, you usually need to declare them at the beginning of your program. However, since Python is a completely object oriented programming language, there is no need to declare variables (all variables are objects).

The webpage <https://www.learnpython.org/en/Variables_and_Types> has a nice tutorial on variables (use a Chrome browser). Below are various examples of variables  and :

Variables assigned to integers:





Variables assigned to real numbers (floating point numbers):





Variables assigned to character strings:





We can also make a list of variables. Below is a list assigned to the variable :

*d* = [1, 15.3, “Hello”]

You can refer to an element in the list by specifying an *index number*:



print()

🡪 1

retrieves the 1st element in the list, 1, and assigns it to the variable *x*. To retrieve the 3rd element in the list do:



print()

🡪 Hello

Note, the 1st element of the list starts with the index “0” (not the index “1”).

Do the tutorial in the webpage: <https://www.learnpython.org/en/Variables_and_Types>. I found that the tutorial works using the Chrome browser. Notice in the tutorial that adding, subtracting, multiplying use the same operations we are used to:

The following code when executed in Phyton yields 16.3:







print()

🡪 16.3

The following code when executed in Phyton yields “Hello World”:







print()

🡪 Hello World

For strings, the “+” operator operates as a *concatenation* operator.

**Loops & Debugging**

The best way to see how loops work is to examine them using the debugger. First, create a Python project called *Loop.py*. Inside this program put the following code:

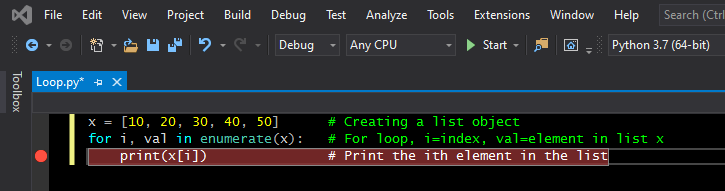
x = [10, 20, 30, 40, 50] # Creating a list object

for i, val in enumerate(x): # For loop, i=index, val=element in list x

print(x[i]) # Print the ith element in the list

The variable  is a list object containing 5 elements. The *for* loop works as follows: when the program gets to the 2nd line of the code, the index  is set to 0 and  is set to 10 (since ). Then, when the program gets to the 3rd line, it prints  which has the value of 10 since . Then the program goes back to the 2nd line, sets  to 1 and  to 20 (since ). Then when it gets to the 3rd line it prints  which has the value of 20 since . Then the program goes back to the 2nd line and sets  to 2 and so forth iteratively setting the index  to 0, 1, 2, 3, 4 until all the elements in the list  are printed. This is the function of a loop: iteratively repeating calculations or functions until a flag tells it to stop (in this case the “flag” is provided by the internal Python function *enumerate*—it knows when it reaches the end of the list).

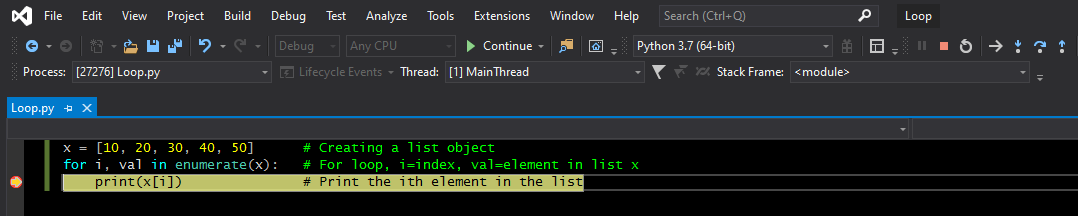
Lets examine how the *for* loop works using the debugger. First, set a *breakpoint* at the 3rd line of the code by either clicking your mouse in the border region of the window or by going to DebugNew Breakpoint in the menu.



Loops must be indented after the *for* statement. Indentation lets Python know you are making a *for* loop.

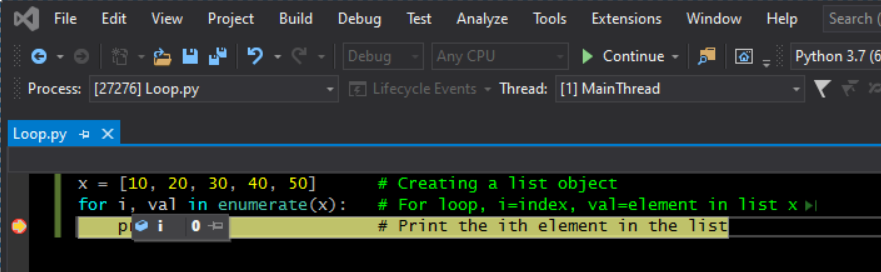
*breakpoint*

After making the *breakpoint*, run the code by pressing the green arrow Start button. The program will halt at the *breakpoint*.



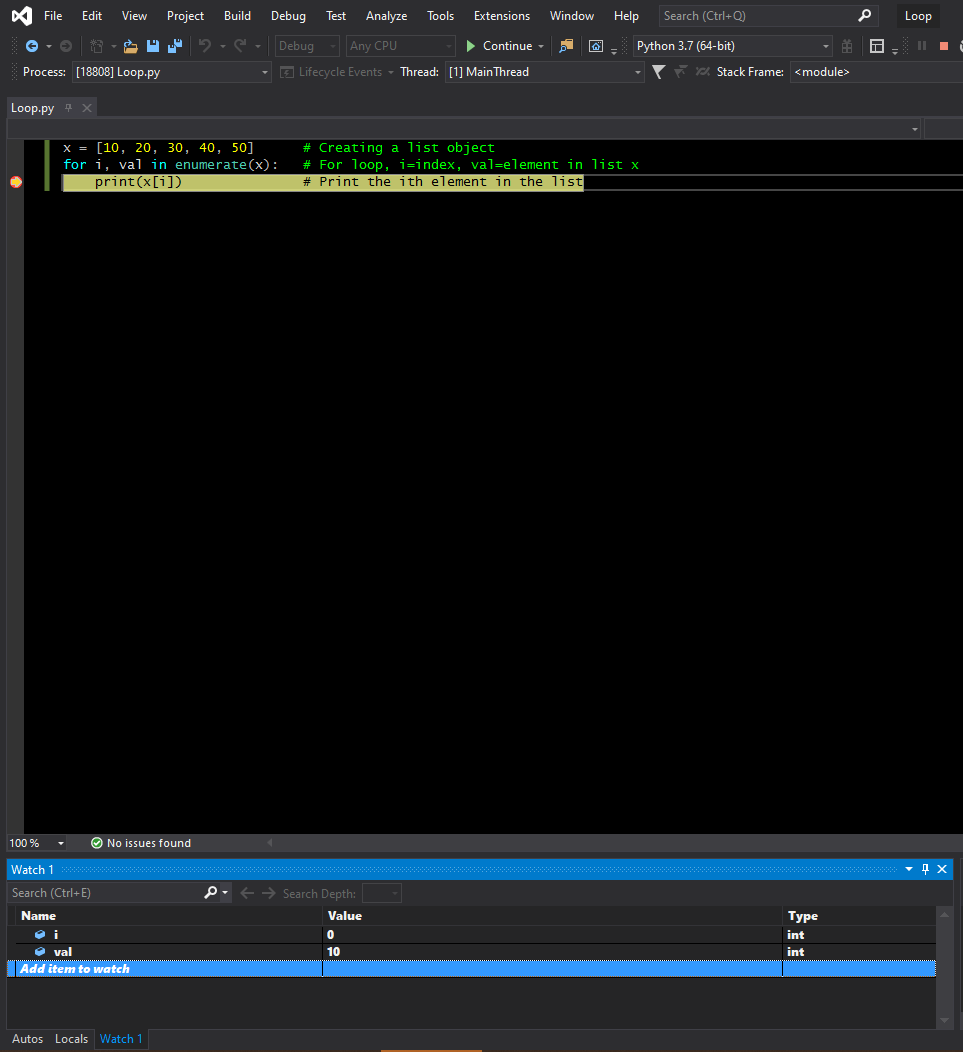
The Step Into button allows one to step through the code line by line.

By pressing the Step Into button (or going to DebugStep Into in the menu), you will be able to step through the code line by line. By hovering your cursor over a variable, you will be able to see the value assigned to the variable (see below, we see that initially that —hovering over  shows that it is 10).



When you 1st press Step Into, you will see that the program goes back to the 2nd line (and  is still 0). Press Step Into and the program steps to the 3rd line and sets  to 1 and  to 20. Pressing Step Into iteratively and you will see  increment from 0 to 4.

Another way to see the values of variables is to use the Watch window (see below).



Type  and  into the Watch fields to see their values

Watch monitors values of variables.

Now when you press Step Into, you see the variables change dynamically.

**Reading Data Files**

This section will show how to read data from files into arrays. Note that the variable  in the last example was a list of objects, not an array of objects. One cannot multiply lists like you can do for arrays (or matrices). Unfortunately, Python 3.7 does not have built-in support for arrays. We must install a module that does, such as *numpy*. Install *numpy* by using *pip* (see Python Tutorial #1):

py -m pip install Numpy

*Numpy* is a Python module containing a library of high-level mathematical operations for multi-dimensional arrays and matrices. It contains all the support scientists and engineers need for arrays and matrices. To use *numpy*, you will need to import it with the line:

import numpy as np

The designation *np* is a short name alias for *numpy*. To create the array  (instead of the list  used in the last section), we would write the code

x = np.array([10, 20, 30, 40, 50])

The designation *np* is attached to the name *array* to inform Phyton to look in the *numpy* library for the *array structure*.

Numpy also has a nice library function for reading data files commonly used by scientists and engineers: *loadtxt*. See documentation on the webpage at:

<https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html>

for more on the *loadtxt* routine. The webpage <https://numpy.org/> is also a good place to find out more about other *numpy* library routines.

**Finding the Mean of a Set of Grades**

We are now ready to read in Bevington’s grades file called *grades.txt*. Create a Python project called *Grades.py* and insert the following code:

import numpy as np # the alias for "numpy" will be "np"

x, y = np.loadtxt('grades.txt', unpack=True) # unpack=True transposes columns

print(x) # of data into rows of data and

print(y) # places it into the arrays x and y

Make certain the *grades.txt* file is in the same folder as the *Grades.py* file.

When you run the program you should see that the arrays  and  contain the data in *grades.txt*.

Now we are ready to find the mean of the grades given in Bevington. The formula for the mean is



So, to find the mean we must sum all the data points and divide by the total number of points. Using enumerate in the *for* loop (as in the section on **Loops & Debugging**), insert the following code in your *Grades.py* code.

import numpy as np # the alias for "numpy" will be "np"

x, y = np.loadtxt('grades.txt', unpack=True) # unpack=True transposes columns

print(x) # of data into rows of data and

print(y) # places it into the arrays x and y

Sum = 0.0 # initial value of Sum is zero

for i, val in enumerate(y): # For loop, i=index, val=element in array y

Sum = Sum + y[i] # sum up all elements (grades) in array y

print(y[i]) # Print the ith element in the array y

Mean = Sum/(i+1) # Calculate the Mean of the Grades data

print(Mean) # Print the Mean

You do not need the print(x), print(y), or the print(y[i]) statements—I left them in for debugging purposes (you can comment them out with the # symbol). Notice that we had to divide the sum with (i+1) since the i index starts at 0 for enumerate and for arrays in general (not at 1 as given in the formula above for the mean).



Notice how Visual Studio outlines the *for* loop so that you can clearly see it.

**Homework**

Extend the *Grades.py* code to also calculate and print out the standard deviation



And the standard error of the mean



Upload to Blackboard your source code (\*.py) of your Python programs *Loop.py* and *Grades.py*. You will see an assignment on Blackboard called **Python Tutorial #2**.