Python – Lesson 1

Reading

I recommend reading the textbook material first. The lesson notes are to supplement the information found in the textbook.

Read parts of Chapter 1 – for example if running Windows pages 3 and 4, and then from bottom of page 10 till the end of the chapter pertains to you. Read the entire Chapter 2

Make sure to take a look at the Python Keywords and Built-in Functions section at the bottom of page 489 and page 490. You do not need to know right away what all those mean or do, but keep in mind they have a special meaning in Python and therefore you should not use them for any other purpose – such as naming a variable.

Installation

If you do not already have Python installed, you will need to install it. As of the textbook writing the current version was 3.5, and as of the writing of these notes it is already at 3.6. Make sure you install version 3.x. If you have one installed, you do not need to install the latest version. It is recommended you stay with version 3.5 or 3.6.

Writing code in the interpreter vs. the editor

As the book mentions you can write code directly in a Python interpreter. It is a nice way to try out a line or two of code, but beyond that do not use the interpreter. Write your code in an editor (Geany and IDLE are recommended for Windows), and run your code from there. The advantages include being able to correct your code and re-run it easily. This will be especially helpful when you have longer programs you need to test and work with. It is also easier to share your code, and move it, as you can save it into .py files and transfer them in any

Saving files with a .py extension

Nothing in the text editors forces you to save your files to a file with a .py extension (unlike MS Word, for example that defaults to .doc or .docx). It is highly recommended you add that extension. That does a few things:

1. It lets you and anyone else looking at your files know you are looking at a Python code document
2. It allows the text editors to know what programming language you are using and color codes your code appropriately

3. It becomes very important as you start writing modules (more about that in a different lesson)

Using Variables

If after reading the textbook the concept of a variable still seems abstract, visualize a variable as a box in memory. You give the box a name for example message and you can put in it whatever value you'd like, for example the string "Hello".

```
message = "Hello"
```

When you need to use that variable you simply give Python the variable name (the box name) i.e. message, and it will retrieve what is stored in it, i.e. "Hello".

```
print (message)
```

You can also modify what is in the variable by assigning it a different value, for example "Hello Bob":

```
message = "Hello Bob"
```

or even retrieve what is in it, change it and put it back:

```
message = message + " Smith! "

print (message)
```

If you type in and try all those lines of code the output will be:

```
Hello Bob Smith!
```

Note that printing is not the only thing we can do with variables, and we will use them in many different ways layer, but printing the value of the variable is a quick way to see what is in it.

Variables are never enclosed with quotes. Consider the code from the text book:

```
massage = "Hello Python World! "

print(message)
```

As you know Python will print the string Hello Python World! On the screen.

Now try this:

```
massage = "Hello Python World! "

print("message")
```

This time the printout will read "message". The difference is that we wrapped the word message in quotes, thus making it into a string and not a variable.

Next try this:

```
massage = "Hello Python World! "
```

```
print(Message)
```

You will see that this example ends up returning an error saying that Message does not exist, even though we defined the variable message before the print statement. Well, the problem is that is not actually true – we set up a variable message, but we tried to print what was in variable Message – Python is case sensitive and message is not the same as Message – those are two distinct variables.

Strings and the Special Meaning of the Backslash

As you saw from reading the textbook chapter the backslash can be used in strings to do special things. For example \t will add a tab, a \n will add a new line. That is very useful, and an easy way to add some formatting to your strings.

It can also cause problems however. Consider having to use a path to a file in your code. Let's say you only want to print it out for the user – try:

```
print ("C:\temp\test")
```

Note the very odd output you get:

```
C:	emp	est
```

That is because \t of temp and \t of test got interpreted as tabs. The good news is there are several ways of getting around this problem – try the following options below:

```
print ("C:\\temp\\test")
```

```
print (r"C:\temp\test")
```

```
print ("C:/temp/test")
```

Any of those will work just fine, albeit the forward slash stays a forward slash, but it will work as a file path.

You will not work with filepaths right away, but do keep this information in mind as we work our way forward.

Inputting Values from the User

Setting variables in your code and doing something with them is only so useful, as usually you want to get some input from the user in order to do something with it. In this lesson we will cover a very simple way of getting information from the user – the input function for that. Try the following code:

```
name = input("Please enter your name: ")
```

```
print ("Nice to meet you " + name)
```

In the first line we are using the input function. The input function will display the string we include in parenthesis, much like the print function, but it will return whatever the user enters. We are storing what the user returns in the name variable.

In the following line we are printing the string “Nice to meet you “ and adding the value stored in the name variable to it.

Switching Variable Types

We can do something similar with numbers:

a = input("Enter a number: ")

b = input("Enter a second number: ")

print (a+b)

But wait, why is the output not actually adding numbers? It is concatenating them instead! The reason is that the input function returns a string, so even though you entered numbers they are treated as strings. In order to fix that we can use the int function, much like the textbook used the str function to convert a number into a string:

```
a = input("Enter a number: ")

b = input("Enter a second number: ")

print (int(a)+int(b))
```

or if you want your code to be able to handle decimals you can use:

```
print (float(a)+float(b))
```

The nicer thing to would be to add some other text to the print statement to let the user know what the output means, so you can change the print statement to this:

```
print ("The sum is: " + str(float(a)+float(b)))
```

In order to concatenate the text and the numerical result we had to convert the result back to a string. We also nested a number of functions, the float function and the addition within the str function, which was nested with an additional string within the print function. All of that is perfectly legal in Python, however you can break this line of code into sections if it makes more sense to you, for example the code below will produce the same result:

```
a = input("Enter a number: ")

b = input("Enter a second number: ")

a = float(a)
```

```
b = float(b)
result = a + b
strResult = str(result)
print ("The sum is: " + strResult)
```

Code Comments

Programing languages allow you to add comments to your code in addition to writing code. Comments are ignored by the programming language when the code is executed. It is very helpful to leave notes in the code for yourself as the original programmer or other programmers that have to look at your code later. Good comments will describe what a section of code does to make it easier to find something. They are helpful in explaining complex logic and make any other notes you deem important. It is usually a good idea to put some comments in the beginning of your code describing in general terms what it does, when and by whom it was created and/or revised.

There are two ways to add comments in Python. If you only want to add one line you would put the pound sign before your comment. Everything after the pound will be ignored by Python. The other way is to add three quotes at the beginning of the comment, type as many lines worth of comments as you'd like and finish your comment with a second set of triple quotes. Here is an example:

```
"""
**********************************************************************
Description: This program inputs two values, converts the values into
             floats, adds them up and reports the sum

Author: Rossana Grzinic

Last Revision: 6/14/2017
**********************************************************************
"""

#Input Section
a = input("Enter a number: ")
b = input("Enter a second number: ")

#Conversion
```

```
a = float(a)

b = float(b)


#Performing addition

result = a + b


#Reporting Result

strResult = str(result)

print ("The sum is: " + strResult)
```

You will see once you copy the code into an editor that the comments are colored differently, thus helping you identify them at a glance.

Do not be afraid of white space in your code. Any lines you leave blank are ignored by Python. Leaving blank lines in certain locations greatly helps the readability of the code.

Closing comments

Writing code is partially about getting your code working. However it is also important to have the code properly formatted and commented. It is even more important to write efficient code – find the best variable type to use in your problem, do not add any variables you do not need using, use the best performing algorithms, etc.