



JANUARY 11, 2017

GETTING STARTED WITH PYTHON

TOPIC 2

VINCENT A. DINOTO, JR.
JEFFERSON COMMUNITY AND TECHNICAL COLLEGE
Vince.dinoto@kctcs.edu

Contents

Table of Figures	2
Introduction.....	3
Purpose.....	3
Important Concepts	3
Comment.....	4
Variables	5
Figure 2	6
Assignment 2.1	6
List Variable.....	7
Figure 3	7
Print.....	7
Printing only Part of a Variable.....	8
Printing a List	8
Assignment 2.2.....	9
Basic Math	9
Decision Statements.....	10
Assignment 2.3.....	13
For Loop.....	14
Assignment 2.4.....	16
Quiz.....	16
Index	17

Table of Figures

Figure 1: Example of Comments	4
Figure 2: Example of Variables	6
Figure 3: example of List Variables.....	7
Figure 4: Simple Print	7
Figure 5: Printing Part of a Variable	8
Figure 6: Defining of two Lists	8
Figure 7: Printing of an Individual Element of a List	9
Figure 8: Printing of multiple Elements of a List	9
Figure 9: Basic Mathematics	9
Figure 10: If/else Logic	11
Figure 11: If code	11
Figure 12: If/Elif/Else Flow Chart	12
Figure 13: Code of If/Elif/Else Flow Chart	13
Figure 14 For Statement	14
Figure 15: For Statement with a String	15
Figure 16: For Statement Example	15

Introduction

It is assumed that the learner has the skills and understanding as noted in the introduction. This module will provide a general overview of Python scripting. The Python covered in this module will not be related to geospatial technologies, but instead will be a basic understanding of how Python functions and the structure of the scripts. Some of the concepts in this general module will be covered in more depth when they deal specifically with geospatial technology in future modules.

Purpose

Many times in geospatial technology, the same operation is done on multiple data sets, which causes the user to do repetitive operations. In general, this is moving the cursor around, clicking on different buttons, selecting specific files and commands. As part of the operation, manual decisions are required based on the data to be analyzed, many maps will lack functionality, creativity and flexibility. Throughout these technical skills lessons, good naming conventions will be stressed.

1. Use of Python scripts will provide the learner with ways to explore automating multiple processes in a single script. For example, when creating clips using the same polygon file on multiple features like roads, rivers and landmarks, a single line of script can be used for each clip instead of the multiple processes required to manually perform the task. The data content can provide for different mapping solutions without the need to review the results.
2. Analyses can be automated to determine the best way to utilize data sets.
3. The use of decision-making routines based on how a field is displayed or calculated can provide for more dynamic mapped results.
4. Creation of specialized tools for routine operations that are redundant is a very important concept. For example, if sets of counties are merged together and then used as the clipping boundary for statewide data sets, this could be created in a single simple script. Manual input of the counties would be required; the script would merge the counties and make a new polygon, this boundary would clip the information from the statewide data sets. The real power of this type of script can be seen when the same type of operation is required again for another set of counties.

Remember, this course is designed to make the learner proficient in the use of Python for geoprocessing but not to make the learner a Python programmer. This module will explore the operational components of Python and how to format scripts to make them function properly.

Important Concepts

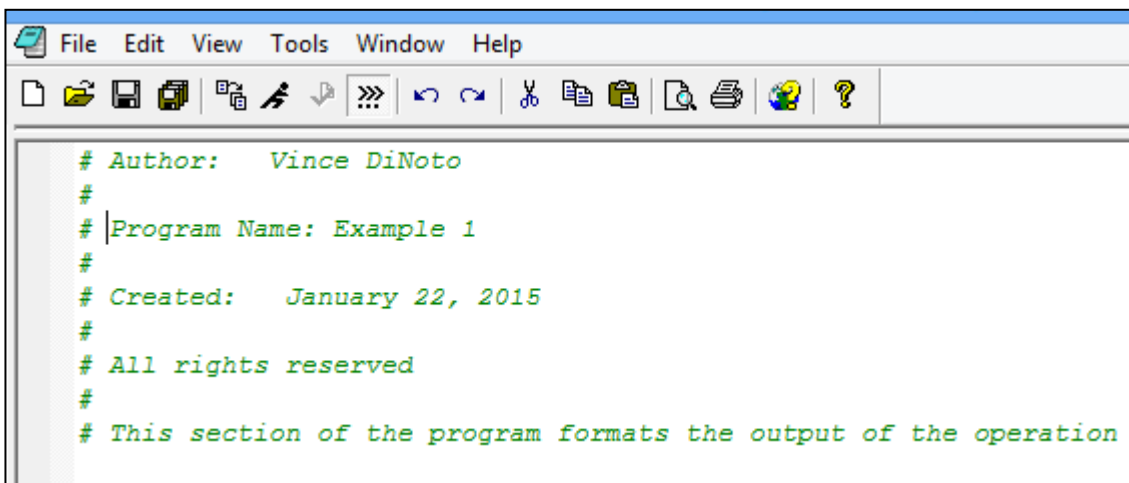
- ☐ Command names are case sensitive and must be used appropriately.
- ☐ Variable names are also case sensitive, for example, *countyName* and *countyname* are not the same variable.
- ☐ In most scripting languages formatting the script is a critical process, this includes the use of indention of lines.

- The concepts of this module will be executed in immediate mode and not in programming mode. When a return (enter) is pressed in immediate mode the scripting line is executed, an exception would be when working with a looping function.

Comment

Any item that follows the pound sign (#) is a comment and will have no effect on the execution of a program line.

- At the beginning of a script the author, organization and the purpose of the script should be defined. This is important for both the author and those that the script might be shared with, the author might have forgotten what the script's purpose was and thus some simple metadata could save time.
- Comments should be used to name the program operation, but also used to separate parts of the program and what their functional purpose is, this makes for easier troubleshooting and modifications. The name of the program or section of the program should be descriptive, there are no character limitations.
- Many times when code is written on a multiple step program, there are errors and the author may not know where the actual problem(s) are located. Placing a comment tag in front of a line of operational code will eliminate the code from the operation, but not delete the line. If the program functions without the line of code, then this is the location of the problem, which can be edited to solve the issue. Multiple lines of code can be commented out (pound sign placed in front of each line).
- A comment can be placed on a blank line as a separator. No text is required after the pound sign so it can also be used to break up sections of the program.
- A comment can be placed after a functional line of code to explain the purpose of the code on that specific line.

A screenshot of a text editor window with a menu bar (File, Edit, View, Tools, Window, Help) and a toolbar. The editor contains several lines of text, each preceded by a pound sign (#), representing comments. The text is as follows:

```
# Author: Vince DiNoto
#
# Program Name: Example 1
#
# Created: January 22, 2015
#
# All rights reserved
#
# This section of the program formats the output of the operation
```

Figure 1: Example of Comments

The comments shown in Figure 1 have the following purposes.

- ❑ The first line of code is used to give the name of the author. Additional information about the author should be given as well. The information should be separated into individual lines. The information might include an email address, phone number, organization name, a storage location.
- ❑ Note the next line is blank since it contains only a pound sign.
- ❑ The name of the program is a critical component, the date the script was created (written) and/or modified is also important.
- ❑ The user rights of someone wanting to use a particular script is also a critical concern. It is suggested that you allow others to use your scripts since this is part of an open learning community. It is important that you know your employer policies on making scripts available to individuals outside of the organization. It is suggested when possible to make the script have a Creative Commons notation.
- ❑ The last line might be placed in the body of the program to explain what is being accomplished in that specific location in the script. Programs may include hundreds of lines of text; none created for this course will be of this size.
- ❑ By using comment statements, it will be easier to edit and modify your script. Many times scripts are created, used and not used again for several weeks or months; the comments will make it possible to understand the functional purpose of the script.

Variables

Variables are place holders for information, the information might be in one of two main types; text or numeric.

- ❑ Variable names cannot use special characters, which includes spaces.
- ❑ A string or text must be placed between a single or double quotation mark. The quotation marks must be used in pairs, so one quote is used at the beginning of the words and another at the end. If single quotes are used, then the same type of quote must be used to finish the expression.
- ❑ If the variable is to be a number only numeric characters can be stored in the variable, a number can be text or numeric, but non-numeric characters can only be text. It is very important to understand that text number characters cannot be used in mathematical operations.
- ❑ The designer can use the plus sign to combine variables of the same type. Numerical items and text items cannot be combined together unless the string function is used. The string function (**str**) is used to make the number into a text variable in a specific operation. The **str** must be lower case letters. For example, the format using the **str** function would be:

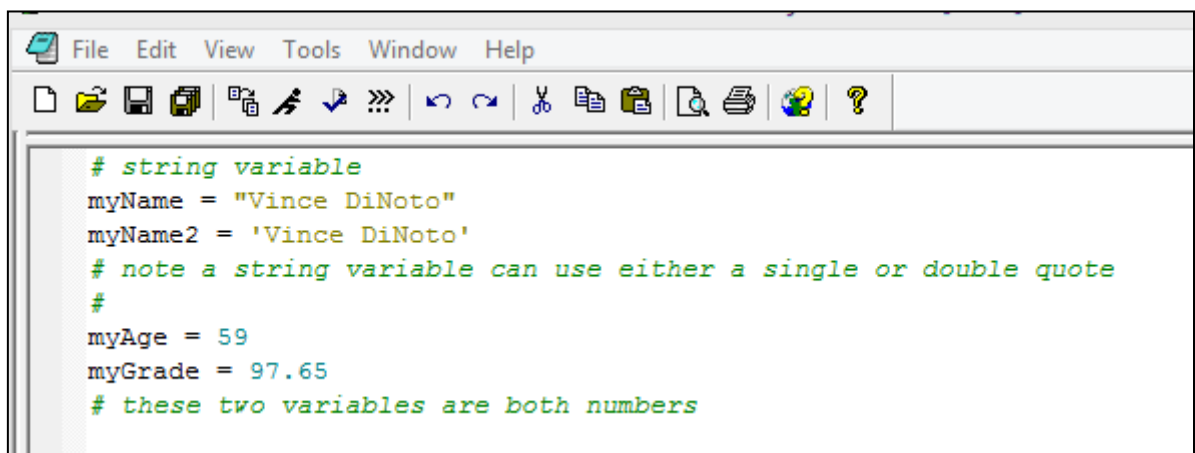
text variable = str(numeric variable)

The text variable and numeric variable are not true variable names since they have spaces in the naming and that is illegal in Python.

- A variable can also be a list, thus populated by more than one element, also known as an array. At this point only lists of one dimension will be considered. A square bracket ([]) is used to make a list.
- While naming the variable it is recommended that both upper and lower case letters be used to make the variable more descriptive. In general, make the first letter of the variable lower case, Python is case sensitive and thus a change in capitalization will create a new variable. Therefore, the string written above might be written like the following as a true variable.

textVariable = str(numericVariable)

The difference is no spaces are used and capitalization has been added to make the name easier to understand.



```

# string variable
myName = "Vince DiNoto"
myName2 = 'Vince DiNoto'
# note a string variable can use either a single or double quote
#
myAge = 59
myGrade = 97.65
# these two variables are both numbers

```

Figure 2: Example of Variables

Figure 2

- The comment line has been previously defined and they will not be discussed again, but notice how the comments are used along with the functional components.
- **myName** and **myName2** are both string variables with the same content, each using a different method to denote the string function.
- **myAge** and **myGrade** are both numeric variables, one being a whole number and the other a decimal, no specification on the number of digits is required.

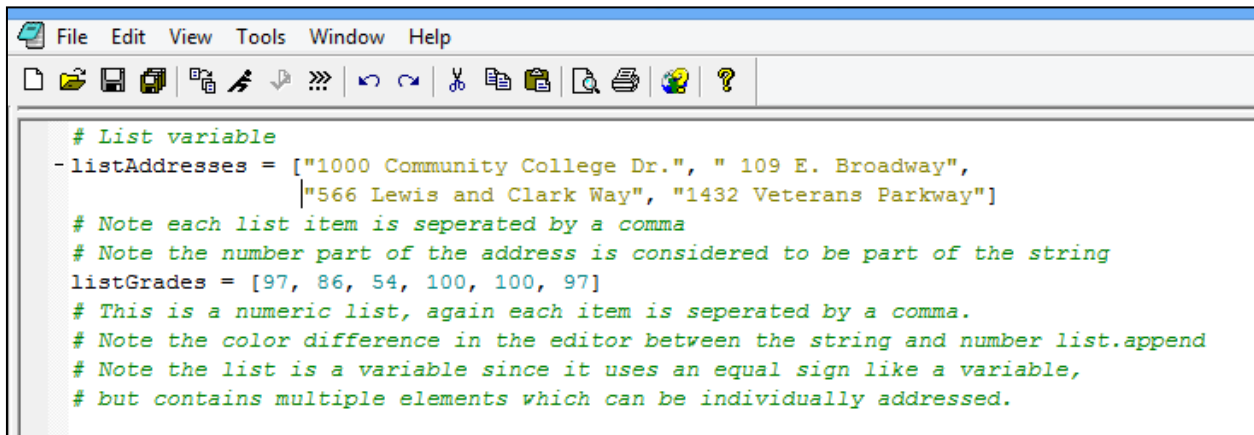
Assignment 2.1

- Create a variable for your first name and one for your last name.
- Create a variable for your age.
- Place comments in the script to define the author, when created and the purpose of the script. Provide the instructor a screen image showing the definitions of these items and the comments.
- It is assumed that the learner can construct simple variables and comments on any script they create, comments are an assumed part of all content required for the entire course.



List Variable

A list variable contains more than one item and all items must be of the same type, such as all string or all numeric. It is suggested that the naming of the list variable use a format with a name so that the user knows it is a list, for example, **addressList** or **listAddress**. As the user becomes more familiar with the operational components of Python, this may become less important, but initially it can be extremely valuable.

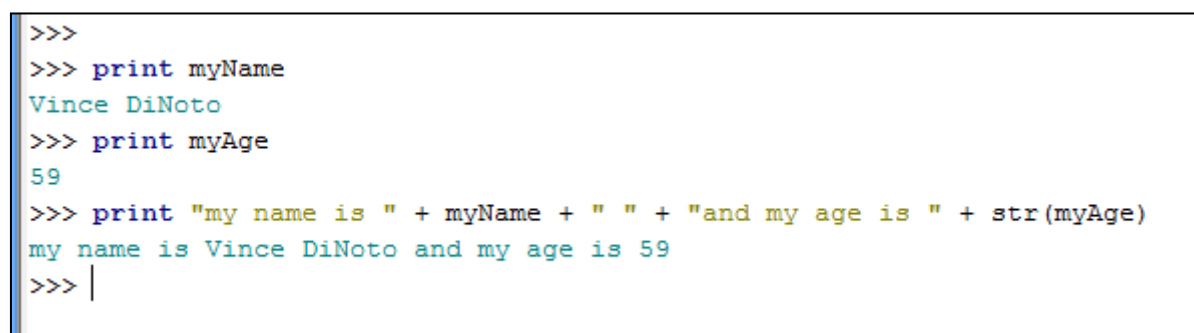
A screenshot of a code editor window with a menu bar (File, Edit, View, Tools, Window, Help) and a toolbar. The code defines two list variables: 'listAddresses' containing three string elements and 'listGrades' containing five numeric elements. Comments explain list syntax and variable naming.

```
# List variable
-listAddresses = ["1000 Community College Dr.", " 109 E. Broadway",
                 "566 Lewis and Clark Way", "1432 Veterans Parkway"]
# Note each list item is separated by a comma
# Note the number part of the address is considered to be part of the string
listGrades = [97, 86, 54, 100, 100, 97]
# This is a numeric list, again each item is separated by a comma.
# Note the color difference in the editor between the string and number list.append
# Note the list is a variable since it uses an equal sign like a variable,
# but contains multiple elements which can be individually addressed.
```

Figure 3: example of List Variables

Figure 3

- There are two defined lists, **listAddresses** and **listGrades**.
- The first list is a string; note the quotes around the numbers and the text. The list starts with a square bracket and items in the list are separate by commas. Note the text is in a different color, not all editors will change the text colors due to variable type.
- In the numeric list, all elements are separated by commas but there are no quotation marks used. The square brackets are used again and the numbers are a different color.

A screenshot of a Python interpreter window showing a series of commands and their outputs. The commands print the values of 'myName' and 'myAge', and then concatenate them into a single string.

```
>>>
>>> print myName
Vince DiNoto
>>> print myAge
59
>>> print "my name is " + myName + " " + "and my age is " + str(myAge)
my name is Vince DiNoto and my age is 59
>>> |
```

Figure 4: Simple Print

Print

- In Figure 4, the first print statement is printing the previously defined variable **myName** and since in immediate mode this information from the previous line is still available. The same is done with the variable **myAge**.

- These two print statements are used to show the results of printing a string variable and a numeric variable.
- When using a print statement all variables must be the same type. In the third print statement, variables are combined together to give a line of text. First text is directly inputted without being part of a variable, next the plus sign and the variable *myName* is inputted. Quotes are used again between the two plus signs to create a blank space. After the third plus sign, additional text is inputted and then finally the *myAge* variable will be interpreted as a string to complete the printed statement. The final displayed line is the result of this combination. Without the string (**str**) command an error would have been generated. The values of the variables are still available from the previous lines executed, this will be true unless the variable is redefined or the editor is closed.

Printing only Part of a Variable

- The *myName* variable is still defined in the PythonWin editor.
- In figure 5 the first line of code prints character space 0 to character space 5. The first position is 0 in a variable. The sixth position (number 5) is not printed. Therefore, positions 0, 1, 2, 3, 4 are printed.
- The next print state would print positions 6, 7, 8, 9, 10, 11.
- The format uses square brackets with a colon separating the positions.
- The third example is printing only a single character, which is position 6.

```
>>>
>>> print myName [0:5]
Vince
>>> print myName [6:12]
DiNoto
>>> print myName [6]
D
>>> |
```

Figure 5: Printing Part of a Variable

```
>>>
>>> print listAddresses
['1000 Community College Dr.', ' 109 E. Broadway', '566 Lewis and Clark Way', '1432 Veterans Parkway']
>>> print listGrades
[97, 86, 54, 100, 100, 97]
>>> |
```

Figure 6: Defining of two Lists

Printing a List

Two lists were previously defined and are still available in the PythonWin program. In figure 6, the print statement and the variable name outputs the complete list. The first list is a set of addresses and the second list is a set of whole numbers. Note the square brackets around the values in the lists.

```
>>>
>>> print listAddresses[2]
566 Lewis and Clark Way
>>>
```

Figure 7: Printing of an Individual Element of a List

Note in figure 7 instead of just placing the name of the variable after the print statement that the number two was placed after the name surrounded by square brackets. Therefore, the returned result is what was in the third position that is represented by the number 2. When a list variable is queried, it brings all the items of the specified position(s) a simple variable returns individual characters.

In figure 8, the items designated in the square bracket will return the result of position 1 and 2, but will not print position 3. The results included in the square brackets use a similar method as to what was shown with individual variables.

```
>>>
>>> print listAddresses[1:3]
[' 109 E. Broadway', '566 Lewis and Clark Way']
>>>
>>> |
```

Figure 8: Printing of multiple Elements of a List

Assignment 2.2

- ☐ The user will define variables that are string and some that are numeric.
- ☐ There should be at least one list variable created.
- ☐ Use good naming convention.
- ☐ Provide your instructor a screen image of the statement each of the variables including the list variable.
- ☐ Print your first name, last name and age, which should be in three unique variables.
- ☐ Print the second member of a List variable.
- ☐ Make sure comment statements are used to describe the process.



Basic Math

```
>>>
>>> a=1
>>> b=2
>>> c=3
>>>
>>> d=a+b+c
>>> d
6
>>> d=(a+b)*c
>>> d
9
>>> d=(a+b)**c
>>> d
27
>>> |
```

Figure 9: Basic Mathematics

In Figure 9, three different variables are defined, **a**, **b**, and **c**. The variable **d** is equal to the sum of the three variables. After the command is executed, no results are automatically displayed. However, the value of variable **d** is now 6. In immediate mode, the value of the variable can be displayed by typing the variable and pressing return. A simple print statement could have been used, which is the preferred method. In the next example, parentheses are used as well as a multiplication. The order of operation is followed in Python so a different result would be obtained with and without the parentheses. A double asterisk is used to raise an item to a power. Note: the variable **d** has a new value after each operation, while the variables **a**, **b**, and **c** retain their original values.

Decision Statements

All computer languages have some type of a statement, which will create a decision statement that uses Boolean Algebra. How the statements are formatted and work vary between different computer languages. Generally, they are based on a logical true or false. When the statement is true, the program branches in one direction and when false a different set of commands are completed. In this section, only a basic understanding of these types of statement will be explored with more detail in later modules.

Typically, in a flow chart a diamond shaped object is used when a decision is required. The diamond is used since there are four points for interaction, the variables are generally shown going into the top of the diamond, with paths going out the bottom and the sides. One of the more common types of decision is known as an **if** statement. Utilizing logic, a statement is written and if true it branches out of the loop, if false generally another comparison is done, known as an **else if (elif)**, the **else if** is true a branch out occurs and if false either it branches to another **else if** or to an **else** statement which is the catch point for all elements that are not true. The concept is if (a) is true do action 1, **else if** (b) is true do action 2, **else if** (c) is true do action 3, and finally if all results fail do action 4.

When writing these statements, a colon must be placed at the end of the line of the if statement, else if statement and else statement in Python. The colon will cause the next line to indent and not to execute automatically on a return like other statements in immediate mode.

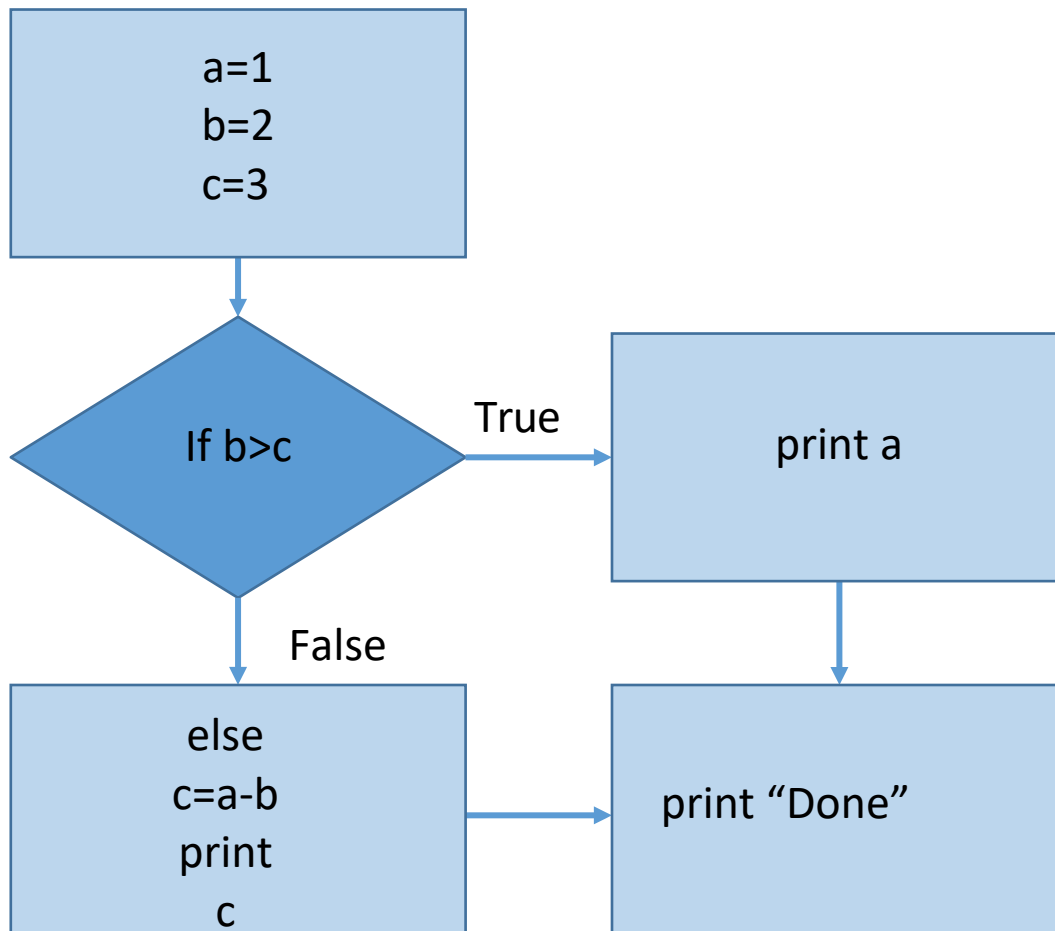


Figure 10: If/else Logic

In Figure 10. The initial variables, a, b, and c are defined. The if statement determines if variable b is greater than variable c. If it is true it would print the value of variable a, but if it is false it does what is in the **else** statement which subtracts the value of b from the value of a, calls it variable c and then prints this value. No matter which path would have been used, it would then print Done. The coding of the logic is shown in Figure 13.

- ☐ The first step is the defining of the variables.
- ☐ The formatting of the if statement, what is indented under the if statement is what is completed if true.
- ☐ The format of the **else** statement, has two items indented under it. Provided the **if** statement is false, what is indented under the **else** statement is done.

```
# define variables
a = 1
b = 2
c = 3
#if statement
- if b > c:
    print a
#Else statement
- else:
    c = a - b
    print c
    print "Done"
```

Figure 11: If code

- Note: the final print statement is back to the original indentation so that no matter which path is taken Done is printed. This example is very simple and in general if statements have more than just two branches.

In Figure 12 an **if/elif/else** statement will depend on the values of a, b, and c.

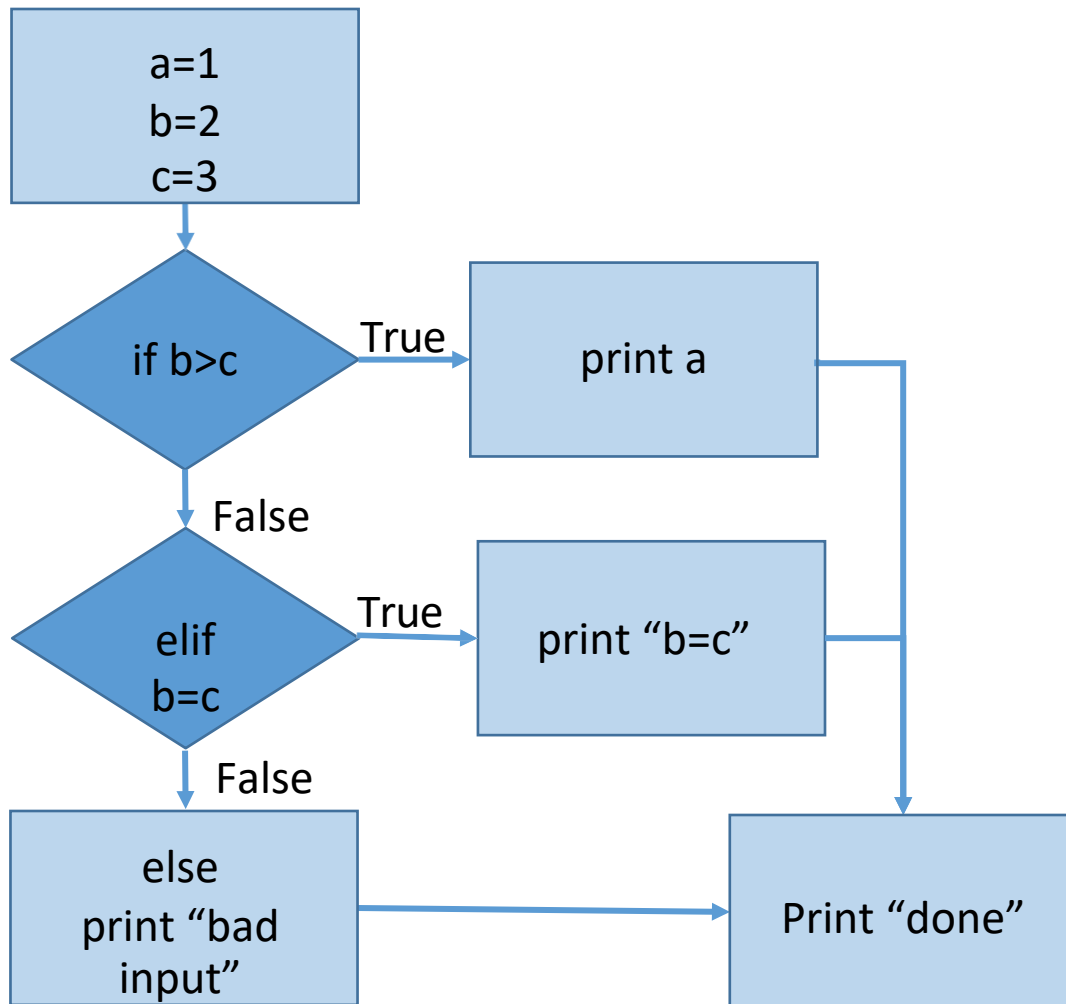


Figure 12: If/Elif/Else Flow Chart

The code in Figure 13 provides a look at the script, for the diagram in figure 12.

- ☐ Three variables are defined, a, b, and c.
- In the **if** statement, provided 'b' is greater than 'c' it is True then the value 'a' is printed, if false the process drops to the elif statement.
- The **elif** (Else If) statement is another if statement and the question is if 'b' is equal to 'c' if true 'b=c' is printed if false the process would drop to the else statement and print 'bad input'.
- No matter which pathway is taken the next operation would be to print 'Done'.
- ☐ In future, code variables would be inputted and not preprogrammed which can have only one result.

```
# define variables
a = 1
b = 2
c = 3
#if statement
- if b > c:
    print a
#else if statement
- elif b == c:
    print "b=c"
#Else statement
- else:
    print "bad input"
#end of program
print "Done"
```

Figure 13: Code of If/Elif/Else Flow Chart

Assignment 2.3

In this assignment, an **if/elif/else** process will be explored.

- ☐ Define four variables
- ☐ Add the first two variables together and multiply the last two variables together.
- If the sum of the first two variables are greater than the product of the last two variables print 'successful'.
- Else if variable 2 is equal to the sum of variable 1, 3 and 4 print 'minor error'.
- Else print 'this program gave a null result'.
- ☐ Print done to show the completion of the statements



For Loop

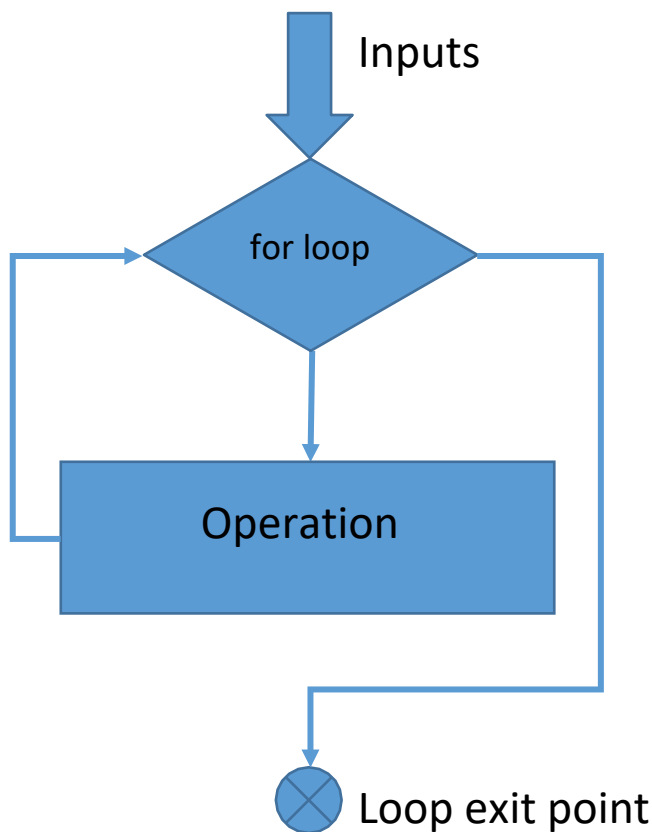


Figure 14 For Statement

Another fundamental concept in most programming languages is the utilization of a looping statement. A looping statement is a redundant function done with different values of variables. A **'for'** loop will go through a process a set number of times and then will exit the loop. The logic is after each process it will check to see if it has met the criteria for the number of operations needed. When these conditions are met it will exit the loop and continue the program. As with the if statement indentions are critical to proper functionality and a colon is used to define the loop. Any processes contained within the indented text will be executed a set number of times before exiting the loop.

Loops can be nested inside other loops, so that the outside loop might go through the process 10 times and the inner loop might be done 5 times, thus the total effect will be 50 operations. As was discovered with the if statement, the use of a colon at the end of the line of the for statement will cause an

indentation required for proper operation. A **'while'** statement is similar to the **'for'** statement and will be covered in the future discussions.

The format of the for statement is:

for variable in range (x,y):

In Figure the left side shows the results of the operation and the right side shows the code.

```

5
5
10
30
120
600
3600
25200
201600
1814400
18144000
199584000
2395008000
31135104000
435891456000
6538371840000
104613949440000
1778437140480000
32011868528640000
608225502044160000
12164510040883200000
255454710858547200000
5620003638888038400000
129260083694424883200000
3102242008666197196800000
77556050216654929920000000
2016457305633028177920000000
54444347252091760803840000000
1524441723058569302507520000000

```

```

test 5
x=5
for y in range(1,30):
    print x
    x=x*y

```

Prior to starting the **for** command the variable *x* is initialized with a value of 5. The value of the *y* variable will initially be 1 (because of the range in the '**for**' statement). The variable will be incremented by one for each time going through the for loop. The loop is only done 29 times since when the value of 30 is reached the loop is exited. Since the print statement is located before the algebraic statement, the initial value of *x* is printed. The first time through the algebraic statement *x* =5 and *y*=1, the next time *x*=5 and

Figure 15: For Statement Example

y=2 which gives *x* a new value of 10, the next multiplication would have *x*=10 and *y* =3. This would continue for 29 times and it is amazing how fast the value of *x* grows.

Instead of a number for the range it could be a string. The range would proceed through the string until all members of the string have been used in the for loop and then would exit the loop. The first line of code in Figure 6 shows the '**for**'

```

PythonWin 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)]
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin'

Current Letter G
Current Letter e
Current Letter o
Current Letter T
Current Letter e
Current Letter c
Current Letter h
Current Letter
Current Letter C
Current Letter e
Current Letter n
Current Letter t
Current Letter e
Current Letter r

```

```

test4
for word in "GeoTech Center":
    print "Current Letter", word

```

Figure 16: For Statement with a String

statement to be used, notice the string is not stored as a variable but instead is directly inputted. Note the colon after the string, which will create the appropriate indentation and everything indented will be completed prior to returning to the **‘for’** statement. The variable **‘word’** will initially have the value of **‘G’**, the print statement will print the string **‘Current Letter’** and then value of the variable **‘word’**. It will return to the **‘for’** statement and move one character to the right thus making the variable **‘word’** equal to **‘e’** and then go through the print statement again.

Assignment 2.4

Create a **for** statement which will write the name of the learner, one character at a time. Initially the first and last name should be in two separate variables. Before each letter is printed there should be text that states that **‘Letter’** and the letter position. There is more than one potential solution to this process and the learner may select the method they would like to use; the only requirement is that a **‘for’** statement must be part of the process.



Quiz

There is a required quiz for this module, which will cover some of the basic Python scripts discussed in this module. The quiz will require that the learner provide images of short pieces of code. Each image needs to use the following naming convention: quiz_question number_last name first initial. For example, Quiz_3_DiNotoV

Index



JCTC_CIT 299_Module 1 _Topic 2_Getting_Started_With_Python by Vincent A. DiNoto, Jr. is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to, accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership. This is an equal opportunity program. Assistive technologies are available upon request and include Voice/TTY (771 or 800-947-6644).