The Python Programming Language

By Constantinos Doskas – ISSA Senior Member, Northern Virginia Chapter

This article discusses the basic building blocks of the Python programming language. It is part of a training series in programming using coding examples that aim to promote the quick learning of developing fun and useful software applications.

Why Python

There are many reasons that people select Python as their first programming language.

One of these reasons is Python's clear and simple syntactical approach. Learning a programming language requires a good amount of effort and dedication. A driving factor for anyone who is involved in programming is the production of results with a minimum amount of effort. Python is the perfect development environment to feed that driving factor and set it in a recursive loop. The more effort the greater the results the greater desire to put in more effort.

Python's advantage over other programming languages is the fact that it is extensible. Therefore, programmers do not have to "re-develop the wheel." There are modules available for any type of programming work. These modules offer "out of the box" tools that allow the programmer to produce results with minimal effort. Examples of the areas supported by Python modules are:

- GUI
- Multimedia
- Image Processing
- Web Frameworks
- Web Scrapping
- Networking
- Databases

There are over 100,000 Python modules available to download, install, and us to help us be productive and complete any project that comes along.

Python is free and available in most operating systems. Python code is processed by the Python interpreter. Because Python is OS independent, the code may be moved from one OS to another and processed with minimal or no modification. Java and C/C++ programmers will find many code similarities with Python and they will be surprised how fast they may produce results with minimal effort/code. If a program has execution time constrains, Cython may be used to translate Python code to C. A Python variant called Jython may be used to compile Python code into Java byte code and make it executable in Java Virtual Machine.

Python and cybersecurity are close knit together. Many cyber tools are either written in Python or have links into Python and are delivered with an embedded Python interpreter.

TIOBE Index and PYPL Index have reported Python as the most popular programming language in the recent years. There is always a link between popularity and stability. Since Python is so popular developers feel comfortable that if they develop in Python their products will be supported for many years to come.

The Python interpreter is available for free from python.org. Python comes with an extensive standard library. In future articles we will describe how additional modules may be downloaded and used.

Building blocks

One of the basic building blocks of any programming language is the *variable.* A variable is computer memory space that is named. Variables are used to store data (symbols, characters, quantities…). In the beginning computers were used to perform calculations. One of the first programming languages was FORTRAN (Formula Translation). Programming languages at that time were used to solve simple equations like x = 2y+25z or much more complex mathematical problems. In the above expression *x,y,* and *z* are variables. Data

is stored in a variable through an assignment operation. For example, the assignment operation x=100 stores the quantity 100 to the memory location named x. The expression x=100 is called a statement.

Variables are categorized in several types, depending of the data that they are designed to store. The basic types of variables are *integer, float, string,* and *Boolean*. Complex variables—*list, tuple,* and *dictionary*—are designed to store data structures that may consist of a variety of data. String variables are designed to store collections of characters. For example :

my_string='It is a beautiful day!'

In the above expression, *my_string* is the variable and the sentence in the single quotation marks is the data stored in it. Note that in an assignment, operation variables are always on the left of the equal sign. Also note that string data is always included in single or double quotation marks. The "pythonic" way prefers single quotation marks. However, when the string data contains an apostrophe then double quotation marks could be used: my_string="That's good." Integer variables and floats are designed to store numbers. Floats are used to hold numbers with decimals such as weights and currency. In the following example *x1* is an integer variable while *y1* is a float:

x1=275

y1=100,000.23

A Boolean variable holds a True or False value. Example:

q1=True

q2=False

Note that a comparison operation always results in a Boolean, True or False, value. Example:

if x==y:

print('The values are equal')

The expression *x==y* (is x equal to y?) results in either True or False. See Appendix A for a list of comparison operators.

One step up from simple variables is the *list* variable. A *list* consists of a collection of data elements. The elements of a *list* may be of the same or different data type. A *list* element may also be another complex data type such as *tuple, dictionary*, … The following variable is a *list* of strings: fruits=['apple','avocado','apricot','fig'].

Note that the elements of a *list* are included in square brackets []. Elements are numbered starting from zero and are referenced by the name of the list followed by the element number included in square brackets. Therefore, to reference the element 'apple,' the following expression is used: fruits[0]; likewise fruits[3] references the element 'fig.'

A *tuple* variable is similar to a *list* variable. The only difference between these two variables is that the elements of a *tuple*, once created, can not be modified. The elements of a *tuple* are included in parenthesis (). For instance, the statement box_size=(8.0,8.0,2.5) creates a *tuple* variable with three numeric values that represent the three dimensions of a specific box.

A *dictionary* variable represents a data structure that contains data that is referenced by a key. An example of a *dictionary* variable is a data structure that contains the basic characteristics of a car. The elements of a *dictionary* variable are included in curly brackets {}.

Each element of a *dictionary* consists of a *key and* a *value* separated by a colon ':' and each element is separated from the adjacent element by a comma. Therefore, the statement that creates a dictionary variable for the characteristics of a car example is:

car={'color' : 'Blue','type' : 'Sedan','make' : 'Toyota', 'model' : 'Camry'}

In the above statement 'color', 'type', 'make', and 'model' are the *keys* and 'Blue', 'Sedan', 'Toyota', and 'Camry' are the corresponding *values*.

Comments in a Python program start with the # character and can be coded in the beginning of a line or after a program statement.

Another basic concept in computer programming is the *function*. Functions represent actions and consist of blocks of code that are reusable. An example of a built -in function in Python is *print*. The print function is used to display data to an output device. The default device is the Python shell that posts data on the computer's monitor. We are going to be using the print function in the following examples.

Finally, Python is an object-oriented platform that can be used to write procedural or object-oriented code. As we will see in future articles, objects consist of sets of variables and functions that are placed in a single block of code and designed to provide specific functionality. The functions that are included in objects are called methods.

Example:

Import time

……..

time.sleep(2)

In the above example time is an object that has a method called sleep. The sleep method takes an argument that is an integer number of seconds. This method is used to halt execution of a program for the amount of seconds passed as argument. In the above example execution of the program will be halted for 2 seconds.

Creating and assigning variables

In figure 1 we have two panes displayed. The pane on the left is the Python editor; the pane on the right is the Python shell. The print function posts the program output to the Python shell. Note the two comment lines in red characters on the top of the program. Also note that "print" and "True" are colored, which identifies them as reserved words. Reserved words cannot be used to create new variables or functions.

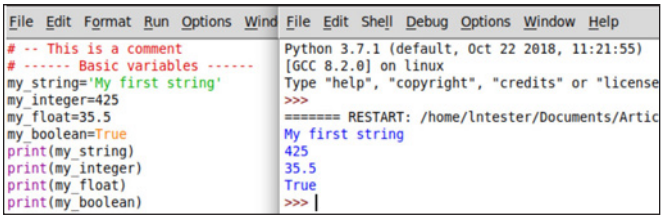

Figure 1 – Creating basic variables and printing their contents

In figure 2 there are *list* variables containing various types of elements. As in the previous example data is displayed on Python's shell by the print function. Note that comments may start at the beginning of the line or after a program statement.

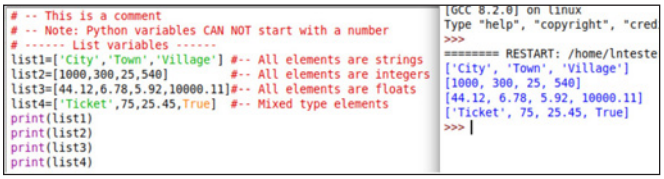

Figure 2 –Creating and printing list variables

In figure 3 a statement is used to manipulate the contents of a *list* element. The index of an element is used to reference the element, [0] is the first *list* element, [1] is the second *list* element…. In the right pane the output before and after element manipulation is displayed.

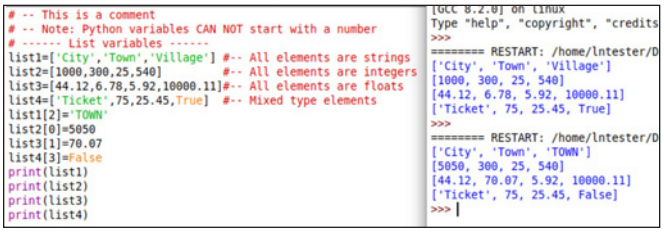

Figure 3 – Manipulating list elements

Note that the same data as in example 1 is now stored in *tuple* variables.

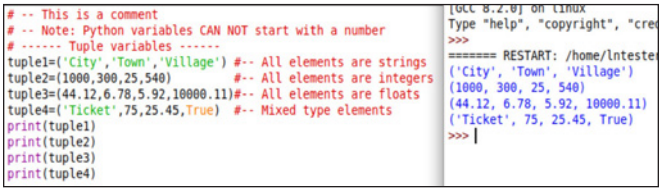

Figure 4 – Operation on tuple variables

Note: Attempting to manipulate a *tuple's* element (tuple[4]= False) results in an error posted in red print on the right pane.

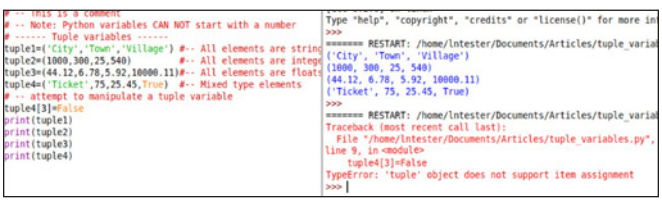

Figure 4b – Tuple element manipulation error

In figure 5, when an unknown *key* is used to reference a *value*, an error is displayed.

Figure 5 – Creating and processing dictionary variables

Flow control using loops

The basic building blocks are used in completing a task or developing a process that manipulates a set of data to produce a result. To develop a process, processing loops are used. Keywords like *while*, *with*, and *for* control the processing of a loop. Loops last as long as a certain condition exists. Variables are used in the creation of conditions.

In figure 6 the *list* variable *ports* is used to store known computer ports. A *for* loop is used to iterate through the data stored in *ports*. Finally, a print statement is used to print the names of those ports.

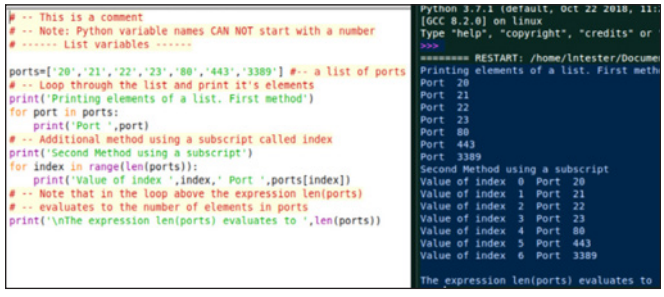

Figure 6 – Creating and processing list variables using a for loop

Using a *for* loop we may also print data that is stored in a *dictionary*. Note that the values stored in a *dictionary* are accessible via the *dictionary keys*. *Dictionary values* may be complex variables (i.e., lists, tuples, or other dictionaries). In figure 6b a *dictionary* has *keys* representing network ports. The value of each *key* is a *tuple* containing two strings: the service that usually runs on the port and the corresponding transport protocol.

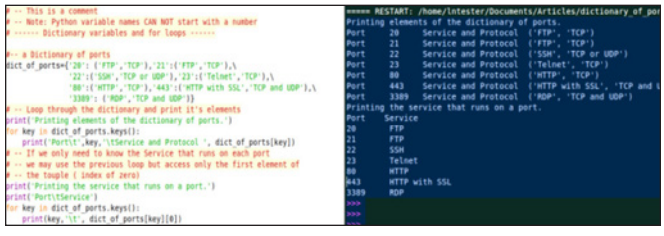

Figure 6b – Creating and processing dictionary variables using a for loop.

Extracting data

In the next example we will see how complex data may be stored in a *dictionary* and how it could be extracted. The data is stored with *keys* representing Windows event numbers. Each Windows event has many pieces of information attached to it. Here we will examine a structure that will hold some of that information and create code to retrieve it. The program will allow a user to enter the event that must be examined.

The program will search the data and report the event category, Windows operating systems that may produce that event, type of event, and a comment related to the event (figure 7).

```
File  Edit  Format  Run  Options  Window  Help
#-- Taining code developed by Constantinos Doskas ISSA-NOVA
# ------ Using an input statement to receive data from a user ------
#-- Dictionary of Windows Events (Sample)
#-- Creating a lookup system of Windows Events
#-- Each key in the dictionary pulls data that is stored
#-- in the form of another dictionary.
```

Figure 7 – Comment section

```
dict_of_WinEvents={'4634': {'Category':'Logoff','Type':'Sucess',\
                            'winOS':['Win2008','Win7','Win8.1',\
                            'Win10','Win2012R2','Win2019'],\
                            'Note':'Logoff is Machine Initiated'},\
                   '4647': {'Category':'Logoff','Type':'Sucess',\
                            'winOS':['Win2008','Win7','Win8.1',\
                            'Win10','Win2012R2','Win2019'],\
                            'Note':'Logoff is User Initiated'},\
                   '4624': {'Category':'Logon','Type':'Sucess',\
                            'winOS':['Win2008','Win7','Win8.1',\
                            'Win10','Win2012R2','Win2019'],\
                            'Note':'Account logged on. Details on: Logon Type t
                   '529' : {'Category':'Logon','Type':'Failure',\
                            'winOS':['WinServer2000','Win2003','WinXP'],\
                            'Note':'Uknown User/Bad Password'},\
                    '530' : {'Category':'Logon','Type':'Failure',\
                             'winOS':['WinServer2000','Win2003','WinXP'],\
                             'Note':'Time restriction violated'},\
                    '4656': {'Category':'Object Access - Registry',\
                             'Type':'Sucess/Failure',\
                             'winOS':['Win2008','Win7','Win8.1',\
                             'Win10','Win2012R2','Win2019'],\
                             'Note':'Request of a handle to an object'},\
                    '4657': {'Category':'Object Access - Registry',\
                             'Type':'Sucess',\
                             'winOS':['Win2008','Win7','Win8.1',\
                             'Win10','Win2012R2','Win2019'],\
                             'Note':'A registry Value was modified'},\
                    '4658': {'Category':'Object Access - Registry',\
                             'Type':'Sucess',\
                             'winOS':['Win2008','Win7','Win8.1',\
                             'Win10','Win2012R2','Win2019'],\
                             'Note':'A handle to an object was closed'},\
                    '517' : {'Category':'System','Type':'Success',\
                             'winOS':['WinServer2000','Win2003','WinXP'],\
                             'Note':'The audit log was cleared'},\
                    '577' : {'Category':'Privilege Use','Type':'Success/Failure',\
                             'winOS':['WinServer2000','Win2003','WinXP'],\
                             'Note':'Privileged Service Called'},\
                    '4650': {'Category':'Logon','Type':'Sucess',\
                             'winOS':['Win2008','Win7','Win8.1',\
                             'Win10','Win2012R2','Win2019'],\
                             'Note':'More detain in Logon Type table'}
                  }
```

Figure 7b – Dictionary creation section

```
# -- Task: Extract from the dictionary data as requested by a user
print('Information available for the following events:\n')
for key in dict_of_WinEvents.keys():
    print(key,' ',end='')
print('\n')
while input('Do you need information on one of the above events? (Y/N)').lower()=='y':
    answer=input('Enter Windows event: ')
    if answer in dict_of_WinEvents.keys():
        for key in dict_of_WinEvents[answer].keys():
            print(key,':\t',dict_of_WinEvents[answer][key])
    else:
        print('Data not available')
print('Thank you, call again.')
```

Figure 7c – Main program section

Note in figure 7c that the *input* statement next to the *while* keyword is using the string's *lower()* method. This method will take the user's response and convert it to lower case characters. Therefore, a *Y* or *y* response will be equally accepted. The "==" double equal characters are used to compare the user's response and the data expected by the program. Figure 7d is the program output.

Building tools

Administrators check system status using a variety of command line interface (CLI) commands. One of the commonly used commands is *ping*, which is used to check the readability of a host and how long it takes to receive a response from the host. The *ping* command is mainly using the ICMP protocol and specifically the ICMP echo and reply message. This com-

```
Information available for the following events:

4634  4647  4624  529  530  4656  4657  4658  517  577  4650

Do you need information on one of the above events? (Y/N)y
Enter Windows event: 517
Category :       System
Type :   Success
winOS :  ['WinServer2000', 'Win2003', 'WinXP']
Note :   The audit log was cleared
Do you need information on one of the above events? (Y/N)Y
Enter Windows event: 4656
Category :       Object Access - Registry
Type :   Sucess/Failure
winOS :  ['Win2008', 'Win7', 'Win8.1', 'Win10', 'Win2012R2', 'Win2019']
Note :   Request of a handle to an object
Do you need information on one of the above events? (Y/N)y
Enter Windows event: 577
Category :       Privilege Use
Type :   Success/Failure
winOS :  ['WinServer2000', 'Win2003', 'WinXP']
Note :   Privileged Service Called
Do you need information on one of the above events? (Y/N)N
Thank you, call again.
```

Figure 7d – Sample run of the program

```
__author__='''

#-----------------------------------------------------
                    ISSA Senior Member
#-----------------------------------------------------

    Constantinos Doskas
    MSIA/Six Sigma
    cdoskas@megabizhost.com
#-----------------------------------------------------
'''
# Importing Modules
import sys
import os
# -- Globals
platforms = {
    'Linux1' : 'Linux',
    'Linux2' : 'Linux',
    'Linux' : 'Linux',
    'Darwin' : 'OS X',
    'Win32' : 'Windows'
    }
# --- Common ip Networks -- Modify to reflect your network
#ip_net='10.1.10.'
#ip_net='192.168.0.'
ip_net='192.168.1.'
# -- Functions
def get_os_platform():
  opsys = sys.platform.capitalize()
  if opsys in platforms.keys():
      opsys=platforms[opsys]
  print('Current platform is: ',opsys)
  return opsys

# Main Function
def main():
print (__author__)
os_platform=get_os_platform()
live_ip=[]
req_timeout=6 # -- max number of packets to send
if os_platform=='Windows':
  for hst in range(0,255):
      the_ip=ip_net+str(hst)
      print('\tIP: ',the_ip)
      os_ping=os.popen('ping {}'.format(the_ip)).read()
      # print(os_ping) # -- uncomment to see detail response
      print('TTL ', os_ping.count('TTL'))
      if os_ping.count('TTL') > 0:
          live_ip.append(the_ip)
elif os_platform=='Linux':
  for hst in range(0,255):
      the_ip=ip_net+str(hst)
      print('\tIP: ',the_ip)
      os_ping=os.popen('ping -c {} {}'.format(str(req_timeout), the_ip)).read()
      #print(os_ping) # -- uncomment to see detail response
      print('TTL ', os_ping.upper().count('TTL'))
      if os_ping.upper().count('TTL') > 0:
          live_ip.append(the_ip)
  #-- Print results
if len(live_ip) > 0:
    print('Accessible IP list\n {}'.format(live_ip))
else:
    print('No network hosts accessible')
return


# --- Processing starts here, then main() is called
if __name__ == '__main__':
    try:
        main()
        print('Process completed. Success')
    except KeyboardInterrupt:
        print ('Exiting Ping Sweep Program] \n\t',\
               'Have a wonderful time')
        sys.exit(0)
    except Exception as e:
        print (e)
```

Figure 8 – Program code

mand is available in most operating systems. UNIX/Linux and Windows versions of *ping* are similar and differ a little on syntax and output format. The command is manually entered in a UNIX/Linux Terminal or a Windows Command Prompt and Powershell windows.

To manually check the readability of multiple hosts is time consuming. A solution is to write a simple Python program that tests all hosts and presents the results, while the administrator works on other projects. Next, we will present a Python program that will accomplish that task. The program can be used to test all hosts of a class C network using IPV4 addresses.

Figure 8 displays the program code.

Here is a sample run of the program (figure 8b).

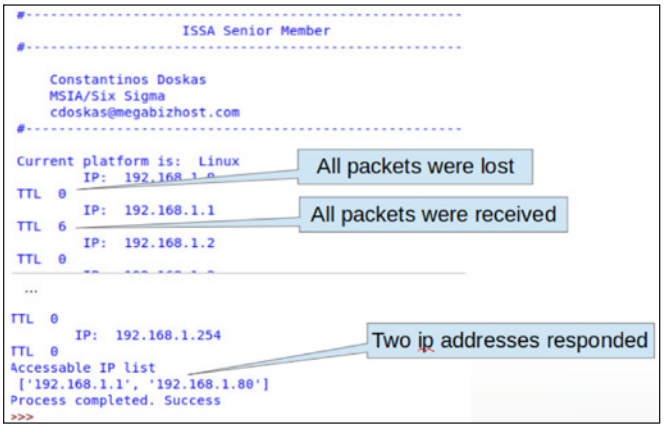

Figure 8b – Program results

Appendix A – Comparison operators

Comparison operators are used to compare two values. A comparison operator is placed between the two values which could be in the form of raw data or data stored in variables. The result of a comparison is either False or True.

COMPARISON OPERATOR TABLE				
		Example		
Operator	Meaning	Data	Comparison	Result
==	Equal To	X=5, Y=3	X==Y	FALSE
!=	Not Equal To	X=8, Y=11	X!=Y	TRUE
>	Greater Than	X=1, Y=6	X>Y	FALSE
<	Less Than	X=100,Y=22	X<Y	FALSE
>=	Greater or Equal	X=50,Y=62	X>=Y	FALSE
<=	Less or Equal	X=40,Y=40	X<=Y	TRUE

Appendix B – Logical operators

Logical operators produce a result based on the value of one or two operands.

LOGICAL OPERATOR TABLE				
		Example		
Operator	Meaning	Data	Operation	Result
and	TRUE if both TRUE	X=TRUE, Y=TRUE	X and Y	TRUE
		X=TRUE, Y=FALSE	X and Y	FALSE
or	TRUE if either TRUE	X=TRUE, Y=TRUE	X and Y	TRUE
		X=TRUE, Y=FALSE	X and Y	FALSE
not	TRUE if FALSE	X=FALSE	not X	TRUE
	FALSE if TRUE	Y=TRUE	not Y	FALSE

Example:

If it is the end of the month **and** the end of the day **then** run the end-of-month procedure.

If temperature is less than 32 **or** temperature is greater than 120 **then** raise an alarm.

If not darkness **then** light

Conclusion – Looking ahead

Python is a popular programming language with clear syntax that can be used to automate our every day tasks. As we will see in future articles, Python may also be used to create powerful applications and tools. Some topics that will be included are data visualization, multi-processing and multi-threading, developing GUI-controlled systems, developing websites, extracting data from the Web, creating encryption engines, analyzing logs, and capturing network traffic.

Python has a long list of libraries available to be used in simplifying development. We will examine many of these libraries (called modules in python) and present working examples of how to use the objects and methods they contain. As we go along with these tutorials the focus will be in developing programs that help cybersecurity professionals to be more productive and innovate new tools.

Source code, installation of Python, and help getting started are available to download at https://www.issa.org/page/JournalResources.

About the Author

Constantinos Doskas is head of the IT and Security Department of Olympus. He has been involved in information systems management and development for over 30 years. He is currently involved in mentoring graduate students and ISSA members in Northern Virginia. Topics include various programming languages and databases. He may be reached at cdoskas@ofdcorp.com.