



Module 3 - System Administration Python

Session 1 - Introduction

Presented by Tim Medin

© SANS, Cyber Aces, Red Siege. All Rights Reserved. Redistribution Prohibited.

YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

Welcome to the Module 3, System Administration. In this sub-section we'll be discussing Python. First, let's get you introduced to this scripting and programming language.

SANS CYBER ACES ONLINE TUTORIALS

YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

1. Introduction to Operating Systems

- 01. Linux
- 02. Windows

2. Networking

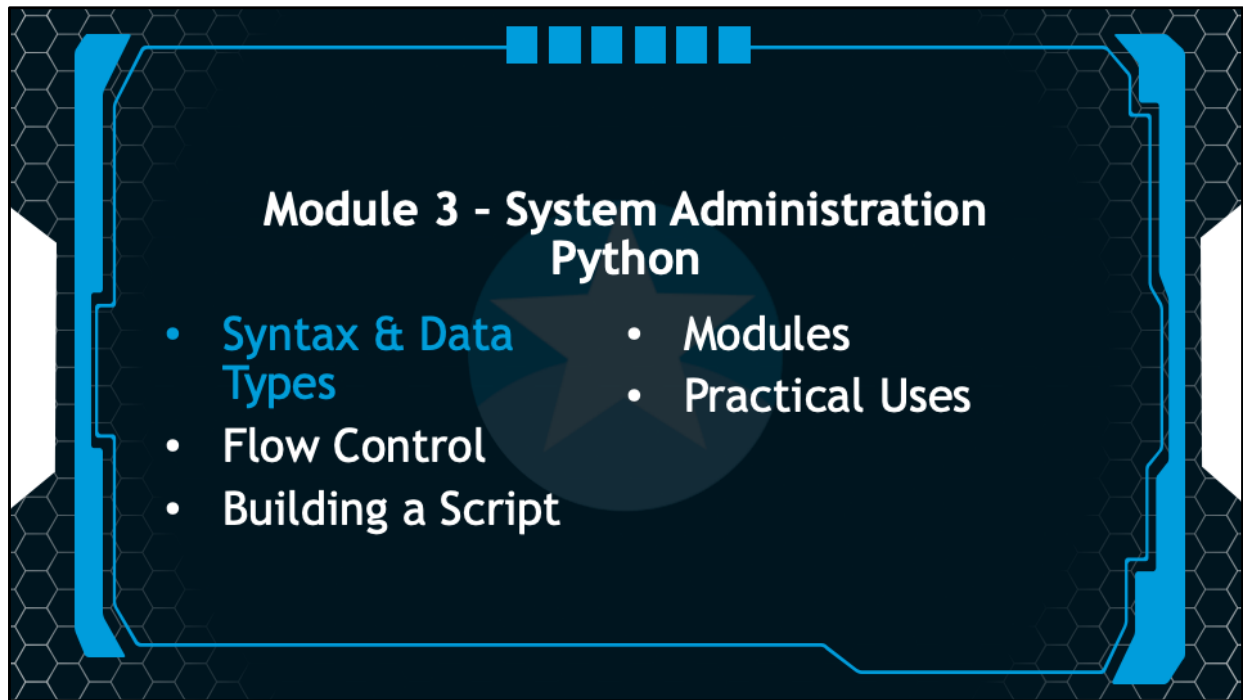
3. System Administration

- 01. Bash
- 02. PowerShell
- 03. Python

This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of system administration and scripting. . This session is part of Module 3, System Administration. This module is split into three sections, Bash, PowerShell, and Python. In this session, we will begin our examination of Python.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

For more information about the Cyber Aces program, please visit the Cyber Aces website at <https://CyberAces.org/>.



Is this section, you will be introduced to Python.



What is Python?



Popular programming language created by Guido van Rossum, and released in 1991

Commonly used for:

- Scripting (many operating systems)
- Web programming (server-side)
- Software development
- Exploit development

Interpreted language, meaning no compilation is required and it is run from text which makes for fast prototyping

Current version is 3. Version 2 is end of life (EOL) 1/1/20

Python is a programming language that was created by Guido van Rossum in 1991. It is a very popular language for scripting as it runs on many operating systems. In fact, many operating systems come with a Python interpreter installed by default (Windows is a notable exception and does not come with a pre-installed interpreter). Python is an interpreted language meaning it does not need to be compiled into native code to be executed. The interpreter takes the text and turns it into byte code which is then executed. Not having to compile the code makes python a language that is great for quick prototyping. Python is commonly used for scripting and automation on Linux and Windows systems. It is also used to run server-side code on web servers. Many organizations use Python to develop software. In addition, Python is commonly used by people in security, especially exploit developers. The current version of Python is 3. Python 2 is end of life (EOL) as of January 1, 2020.



Interpreter



The interpreter runs Python scripts

You can type and execute commands in the interactive interpreter

Installed by default in macOS and most Linux distributions, but a separate install in Windows

Launch interpreter with **python3**, the prompt changes to >>>

We are going to focus on the interactive interpreter for now, but we will write scripts later

```
cyberaces@localhost:~  
File Edit View Search Terminal Help  
[cyberaces@localhost ~]$ python3  
Python 3.6.8 (default, May 21 2019, 23:51:36)  
[GCC 8.2.1 20180905 (Red Hat 8.2.1-3)] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

The interpreter runs the Python scripts. You can also launch the interpreter interactively and run commands right in the interpreter. We are focusing on version 3 and will use the **python3** executable. Python comes preinstall in macOS and most Linux distributions. It is not installed by default in Windows, but there is an installer. We will be using Python in Linux for this reason.

To launch the interpreter you can type **python3** and hit enter. The prompt will change to >>> indicating that you are in the interpreter. We will focus on the interpreter for now. We will write scripts later in this module.



Scripts



On Linux, the first line of the script is the interpreter (remember that from Bash scripting?)

```
#!/usr/bin/env python3
```

This tells Linux to find "python3" using the PATH

Mark the script as executable with:

```
chmod +x scriptname.py
```

This "shebang" will be ignored on Windows

With a proper shebang on Linux you can run the script like this:

```
[cyberaces@localhost ~]$ ./scriptname.py
```

Or you can run it this way (with or without the shebang):

```
[cyberaces@localhost ~]$ python3 scriptname.py
```

To turn our code into a script we need to add a line to the top of the script so when we execute the script Linux can find the correct interpreter. The line looks like this:

```
#!/usr/bin/env python3
```

The "env" executable tells Linux to search it's PATH to find "python3". This line is ignored on Windows. The slight exception is that the Windows Python interpreter will examine any additional options we provide to python3, but in this example we have not presented any.



A note on prompts



We will use the python3 interactive interpreter for most of the exercises

Keep an eye out for the >>> prompt

To launch it, simply run the command below in your Linux VM

```
[cyberaces@localhost ~]$ python3
```

If you want to do the review exercises in script format it can make it easier to edit and debug

For most of the demonstrations and review questions, we will use the python interpreter. The prompt is shown as >>>. To launch the interpreter just type "python3" in your Linux shell.

We will show the interactive interpreter in the slides, but feel free to write a script to solve the review challenges. You can use "gedit" to edit the python scripts and then execute them.



Modules



Modules allow developers to logically organize code

- Python comes with several modules that are very helpful and can save us a lot of development time
- We can write our own modules as well

Modules are loaded with "import"

```
>>> import math
```

```
>>> math.pi
```

```
3.141592653589793
```

Can be loaded into the global namespace

```
>>> from math import *
```

```
>>> pi
```

```
3.141592653589793
```

Additional reading: redsiege.com/modules

Modules can be used to logically organize code. The organization allows us to use different "namespaces" so that objects do not collide names. We can load the modules using "import". For example, the "math" module allows us to perform additional mathematics functions.

We can load the "math" module and get the value for pi.

```
>>> import math
```

```
>>> math.pi
```

```
3.141592653589793
```

You'll notice, that we had to reference the module name. We can load the members into the global namespace (although be careful of name collisions).

```
>>> from math import *
```

```
>>> pi
```

```
3.141592653589793
```

Additional reading: redsiege.com/modules



Print



The `print()` function will output data

```
>>> print(42)  
42
```

We need to use the parenthesis with the print statement

- Version 2 did not require the parenthesis
- This is one of the biggest differences between 2 and 3

In the interpreter you can simply type the variable name and hit enter

The "print" function is how we output data. The function requires that you use parenthesis with the print statement. For example to output the number 42 you'd type this:

```
>>> print(42)  
42
```

Prior to Python version 3, you didn't need to use the parenthesis and could simply type:

```
>>> print 42
```

This change is one of the most significant differences between version 2 and 3.



Variables



Variables do not need to be declared like they do in other languages, such as C

- This makes prototyping much faster
- In the interpreter, you can type the variable name and hit enter to see its value

Variables are dynamically typed

To see the data type use `type(var_name)`

We are going to discuss some of the most common data types later

Unlike other many other languages, especially compiled ones like C and C++, you do not need to declare variables before you use them. You can simply assign the variable and then immediately use it. Of course, if you try to reference the variable before it is assigned then that will cause an error.

Variables in Python are dynamically typed, meaning they are defined as a string or an integer when they are set. To see the type of a variable use `type()`.



Dynamic Typing



To demonstrate dynamic typing, run the following commands in your Python interpreter

```
>>> a = 42
>>> print(a)
42
>>> type(a)
<class 'int'>
>>> a = 'some text'
>>> print(a)
some text
>>> type(a)
<class 'str'>
```

As you can see, the type of the variable "a" changes when we assign it an integer (int) or string (str)

To demonstrate dynamic typing, run the following commands in your Python interpreter

```
>>> a = 42
>>> print(a)
42
>>> type(a)
<class 'int'>
>>> a = 'some text'
>>> print(a)
some text
>>> type(a)
<class 'str'>
```

As you can see, the type of the variable "a" changes when we assign it an integer (int) or string (str)



Quotes



In Python the single quote and double quote are interchangeable, there is no difference

In Bash there is a difference (variable expansion), but not with Python
The code below demonstrates this (Note == is the comparison operator)

```
>>> a = 'Tim'
>>> b = "Tim"
>>> a == b
True
>>> c = 'this is faster to type (no shift key press)'
>>> d = "use double quote if you're using apostrophes"
```

The single quote (') and double quote (") serve exactly the same purpose in Python. Remember, Bash would expand a variable in the double quote but not single quotes, but that is not the case with Python.

The python below demonstrates this (Note that the == is the comparison operator)

```
>>> a = 'Tim'
>>> type(a)
<class 'str'>
>>> b = "Tim"
>>> type(b)
<class 'str'>
>>> a == b
True
```

As we can see above, there is no difference between the single and double quote. Many developers will use the single quotes most of the time since it saves an extra keypress of the shift key.

```
>>> c = 'this is easier to type since I do not have to
use the shift key'
```

If you need the apostrophe (single quote) in your text then it is easier to use the double quotes to define your string.

```
>>> d = "If you're going to use apostrophes, you'll
```

definitely want to use double quotes"



Lists



List is an array of objects and is created with square brackets []

```
>>> weekdays = ['mon', 'tues', 'wed', 'thurs', 'fri']
```

We can access items with their index (zero based) or with ranges

```
>>> weekdays[0]          The first item is zero
'mon'
```

```
>>> weekdays[-2]         Count from the end
'thurs'
```

```
>>> weekdays[2:4]        Range, first number is inclusive, second is exclusive
['wed', 'thurs']
```

```
>>> weekdays[:3]         If first number is omitted it starts at the beginning
['mon', 'tues', 'wed']
```

```
>>> weekdays[3:]         If second number is omitted it goes to the end
['thurs', 'fri']
```

```
>>> weekdays[3] = 'thu'  Assignment
```

Lists is an array of objects and is created with square brackets [].

```
>>> weekdays = ["mon", "tues", "wed", "thurs", "fri"]
```

We can access items in the array by using [indexnumber]. The first item in the list uses and index of zero.

We can count from the end using negative numbers.

```
>>> weekdays[2]
```

```
'thurs'
```

We can use a range by using the start number and an end number (note: negatives work here too). The start number is inclusive, but the end number is exclusive. Another way to say the range is up to, but not including the second number.

```
>>> weekdays[2:4]
```

```
['wed', 'thurs']
```

If the 4 was inclusive, we would see three items (2, 3, 4) instead of just two (2, 3).

We can omit the numbers to start at the beginning or to to the end

```
>>> weekdays[:3]
```

```
['mon', 'tues', 'wed']
```

```
>>> weekdays[3:]
```

```
['thurs', 'fri']
```

We can assign items using the index.

```
>>> weekdays[3] = 'thu'
```



Tuples



A tuple is simply an immutable list

"Immutable" means it cannot be changed, modified, or manipulated

We can reference the objects the same way as lists

Created with parenthesis instead of square brackets

```
>>> weekdays = ('mon', 'tues', 'wed', 'thurs', 'fri')
>>> len weekdays
5
>>> weekdays[3]
'thurs'
>>> weekdays[3] = "thu"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

A tuple is very similar to a list. The big difference is that it is immutable, meaning we can't change modify, or manipulate it. We create a tuple in much the same way that we create a list, but instead of square brackets we use parenthesis. Below, we demonstrate creating a tuple and its immutability.

```
>>> weekdays = ("mon", "tues", "wed", "thurs", "fri")
>>> len weekdays
5
>>> weekdays[3]
'thurs'
>>> weekdays[3] = "thu"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

As you can see, the assignment cause an error.



Float



Decimal numbers of "floating points"

Computers use binary and sometimes that causes issues with conversion to base 10 (how we count)

```
>>> 1 + 2 == 3
```

```
True
```

```
>>> 0.1 + 0.2 == 0.3
```

```
False
```

```
>>> 0.1 + 0.2
```

```
0.30000000000000004
```

Python 3.5 added "isclose" to deal with this issue

```
>>> import math
```

```
>>> math.isclose(0.1 + 0.2, 0.3)
```

```
True
```

The "float" data type is used to hold decimals or "floating point" numbers. Humans count in base 10, but computers use base 2 (ones and zeros). Sometimes this base conversion means we can have unexpected results.

```
>>> 1 + 2 == 3
```

```
True
```

```
>>> 0.1 + 0.2 == 0.3
```

```
False
```

```
>>> 0.1 + 0.2
```

```
0.30000000000000004
```

To deal with this issue, Python 3.5 added "isclose" method to the "math" module (we'll discuss modules shortly).

```
>>> import math
```

```
>>> math.isclose(0.1 + 0.2, 0.3)
```

```
True
```




Review



- What will be the output of the code below?

```
>>> fruits = [ 'apple', 'orange', 'banana', 'grape' ]  
>>> print(fruits[2:3])
```
- Assuming the "math" module contains a method named "sqrt" (square root), will the following code execute successfully or cause an error?

```
>>> import math  
>>> print(sqrt(9))
```

What will be the output of the code below?

```
>>> fruits = [ 'apple', 'orange', 'banana', 'grape' ]  
>>> print(fruits[2:3])
```

Assuming the "math" module contains a method named "sqrt" (square root), will the following code execute successfully or cause an error?

```
>>> import math  
>>> print(sqrt(9))
```



Answers



What will be the output of the code below?

```
>>> fruits = [ 'apple', 'orange', 'banana', 'grape' ]  
>>> print(fruits[2:3])  
['banana']
```

Remember, the first number is the start number and the second is the exclusive end

Assuming the "math" module contains a method named "sqrt" (square root), will the following code execute successfully or cause an error?

```
>>> import math  
>>> print(sqrt(9))
```

▪ This will cause an error. We need to use: `math.sqrt`

What will be the output of the code below?

```
>>> fruits = [ 'apple', 'orange', 'banana', 'grape' ]  
>>> print(fruits[2:3])
```

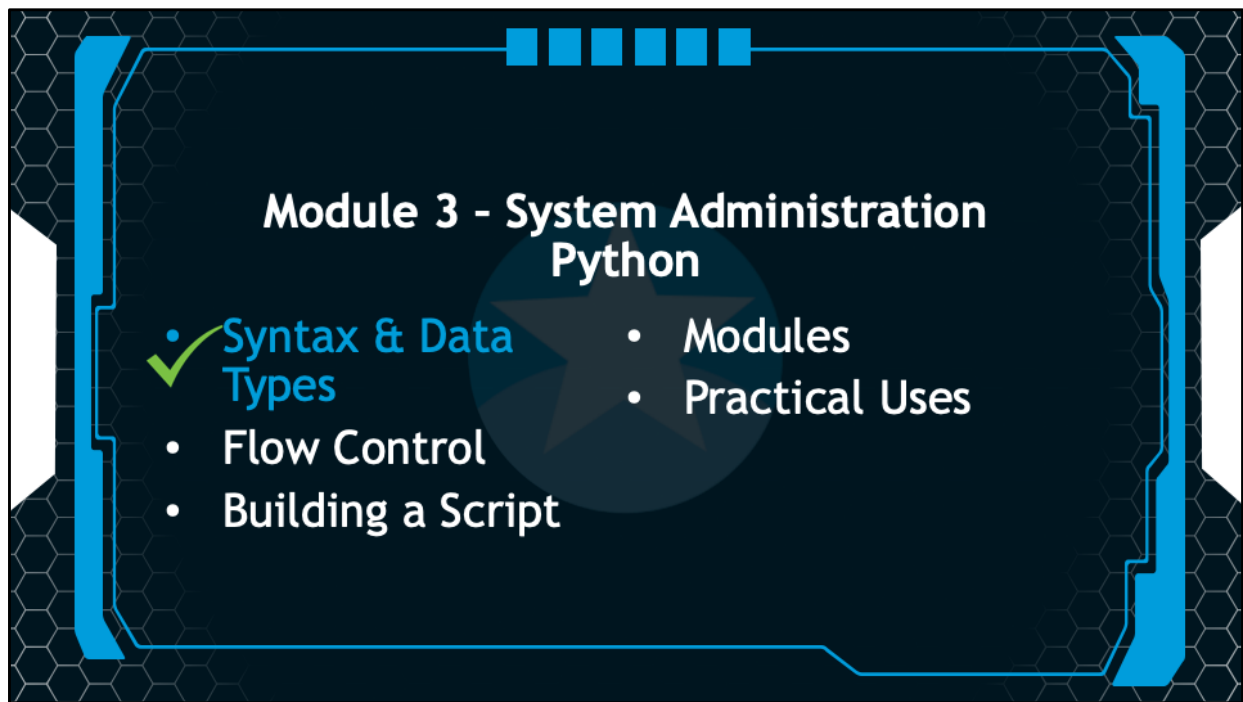
The output will be "banana". Remember, the second number is not included in the output. It will then be the second item up to, but not included the third, which means it will just output the second item.

Assuming the "math" module contains a method named "sqrt" (square root), will the following code execute successfully or cause an error?

```
>>> import math  
>>> print(sqrt(9))
```

This will cause an error since the method is in the "math" namespace and not the global namespace. The code below would work:

```
>>> from math import *  
>>> print(sqrt(9))
```



You've completed the introduction to Python and learned about syntax and data types. In the next session, we will discuss flow control.