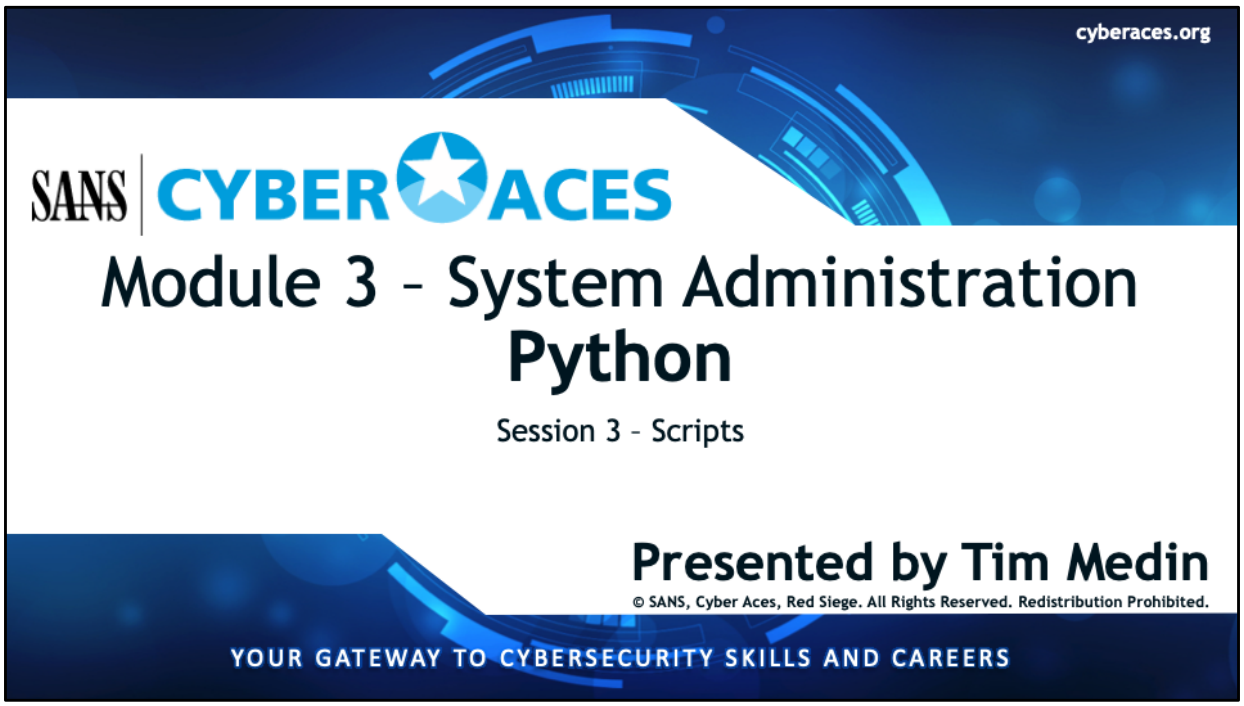
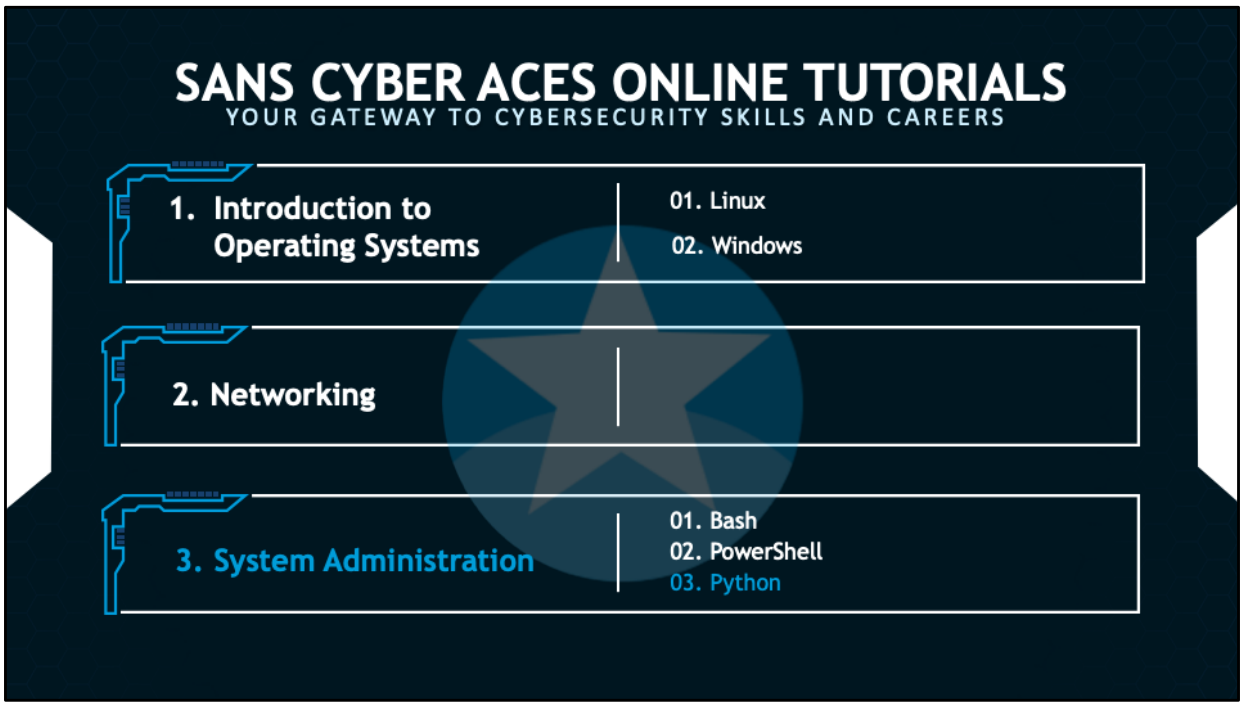


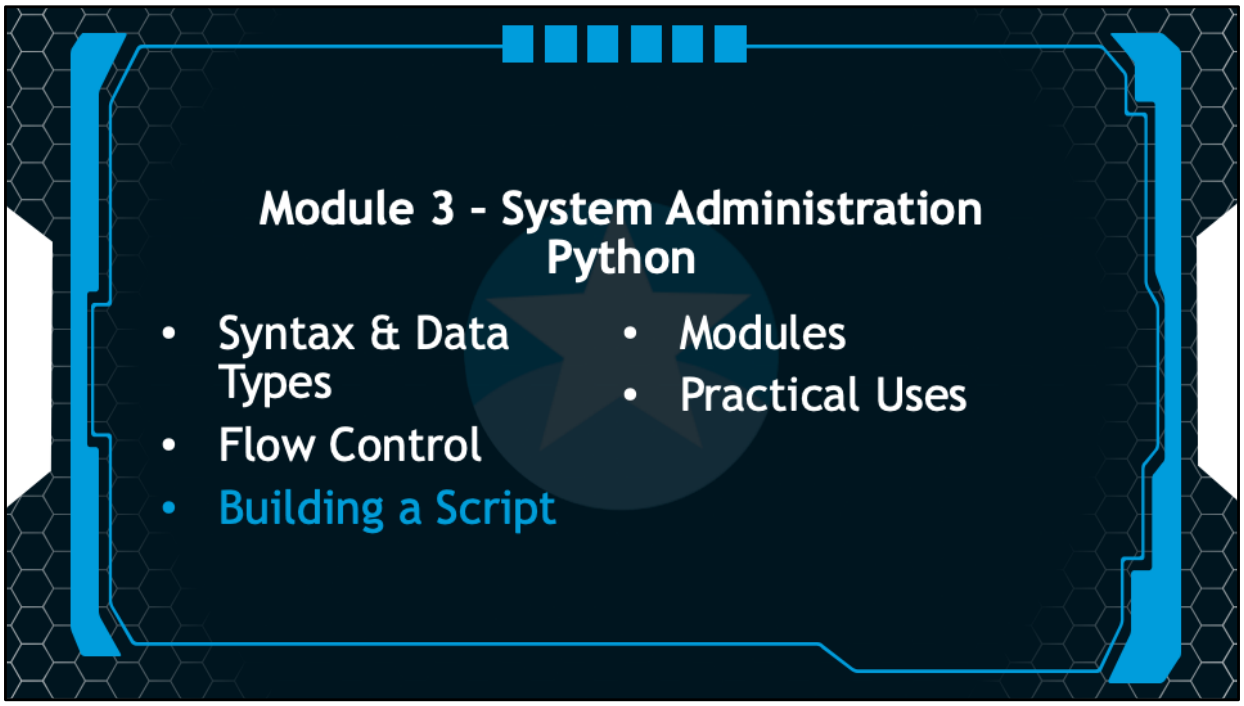
Welcome to the Module 3, System Administration. In this sub-section we'll be discussing Python. First, let's get you introduced to this scripting and programming language.



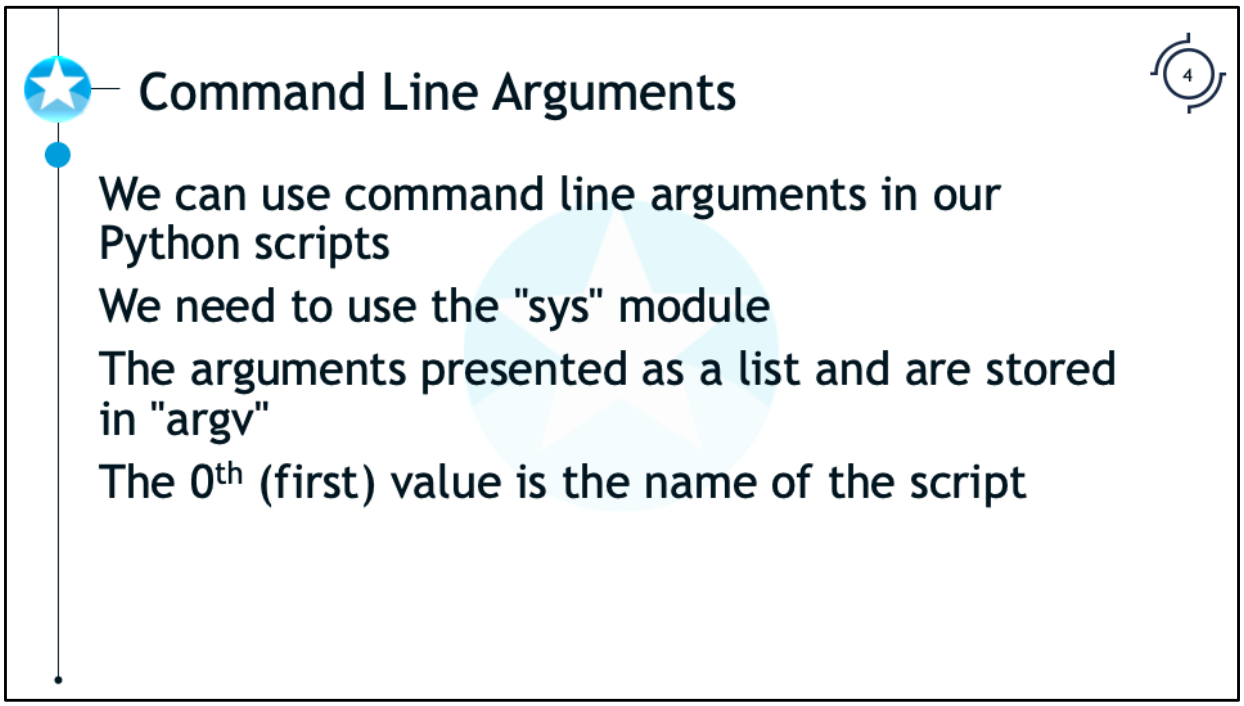
This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of system administration and scripting. . This session is part of Module 3, System Administration. This module is split into three sections,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

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

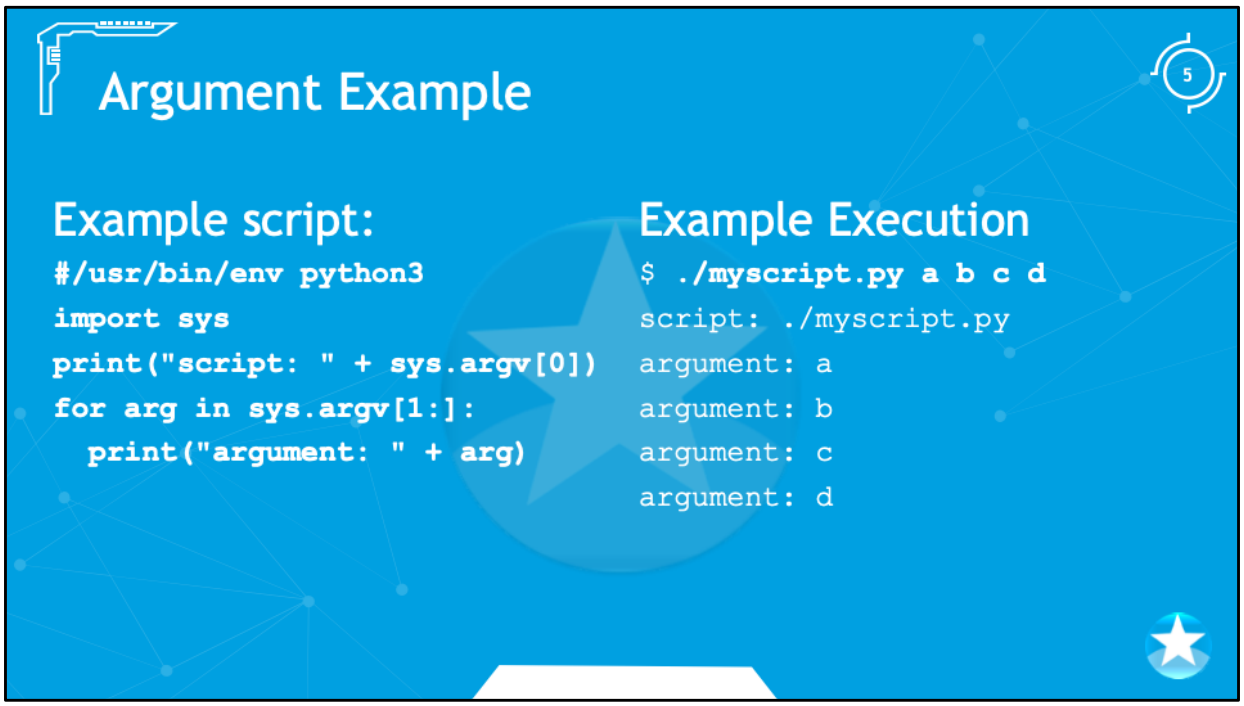
For more information about the Cyber Aces program, please visit the Cyber Aces website at https://CyberAces.org/.



Is this section, you'll learn how to turn your code into a script, functions and using command line arguments.



When writing a script we often want to read command line options presented to our script. To do this, we need to import the "sys" module. The arguments are stored in the "argv" variable. The first item in the list (index -0) is the name of the script. The rest of the items are are the arguments.

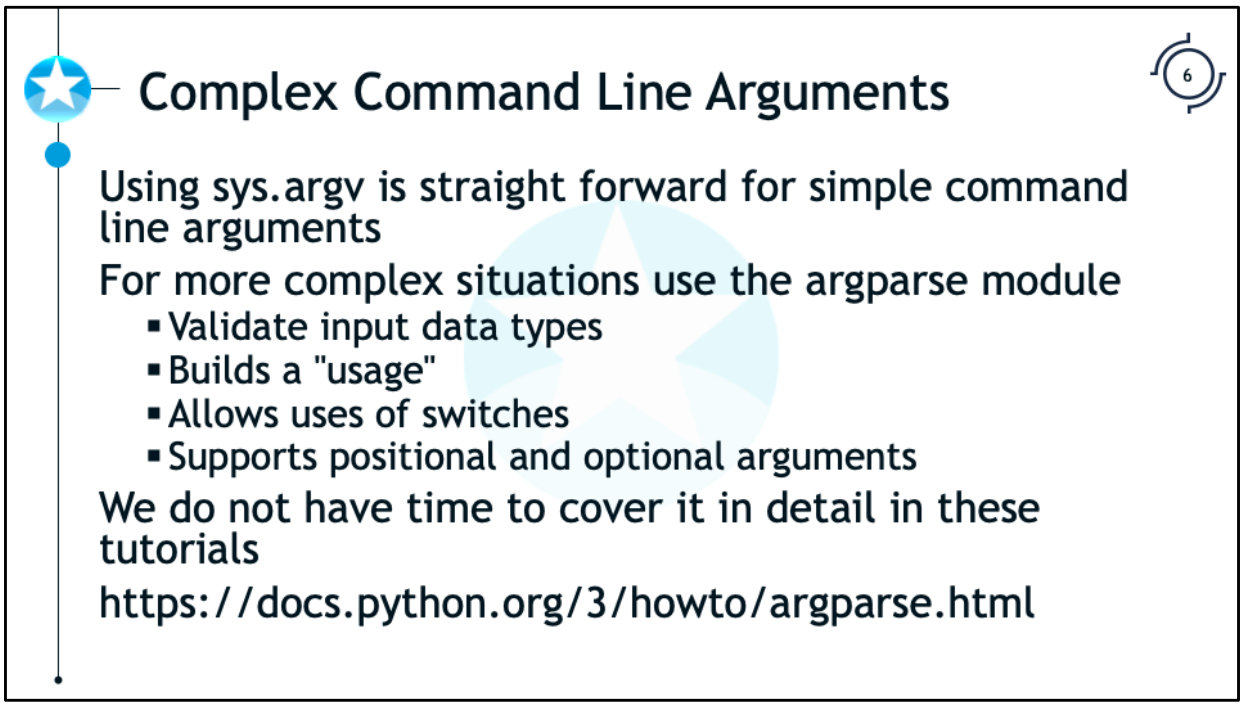


Example script:

```
#/usr/bin/env python3
import sys
print("script: " + sys.argv[0])
for arg in sys.argv[1:]:
  print("argument: " + arg)
```

Example Execution

```
$ ./myscript.py a b c d
script: ./myscript.py
argument: a
argument: b
argument: c
argument: d
```

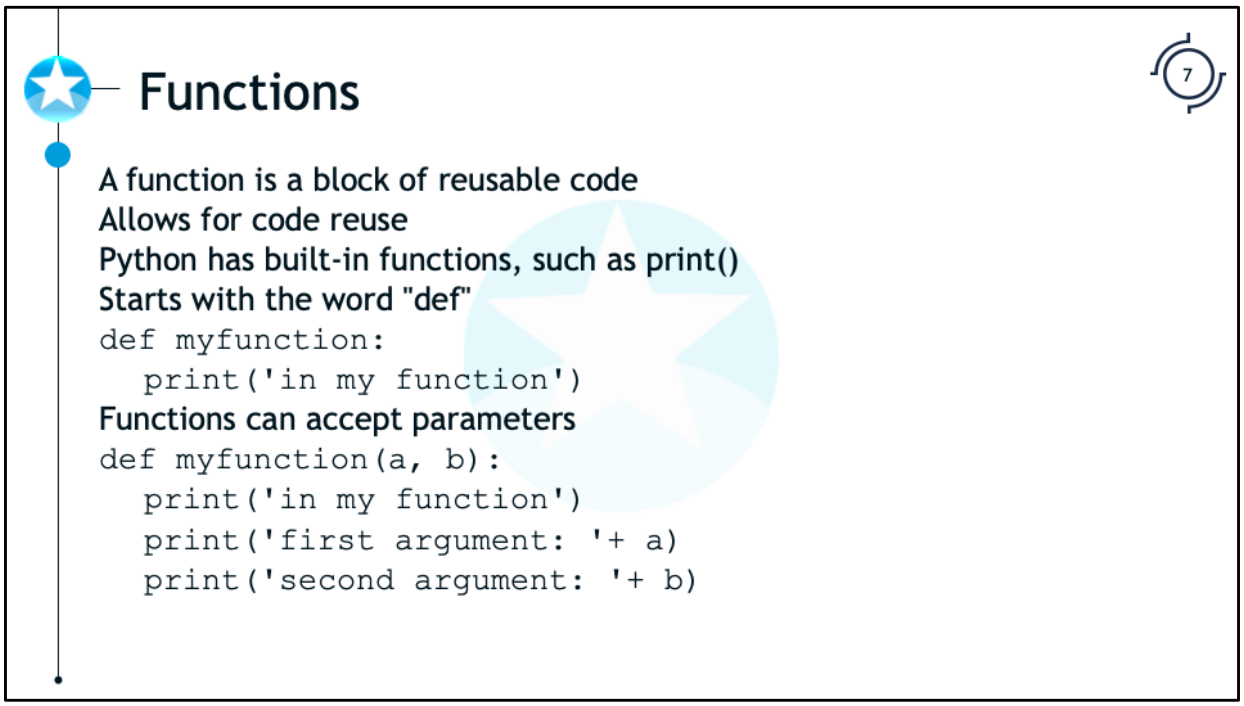


The simple argument parsing with sys.argv works fine in most cases. There will be situations where we need more complex options and choices. Fortunately, the "argparse" module makes this complex usage much simpler. It can

- Validate input data types
- Builds a "usage"
- Allows uses of switches
- Supports positional and optional arguments

We don't have the time to go through all of the features of this module. To read more about this module visit the link below:

https://docs.python.org/3/howto/argparse.html



We can organize reusable code in functions. This allows us call these functions over and over again. Python has built-in functions, such as print(). Functions begin with "def". Below is a function that doesn't take any arguments (not the colon):

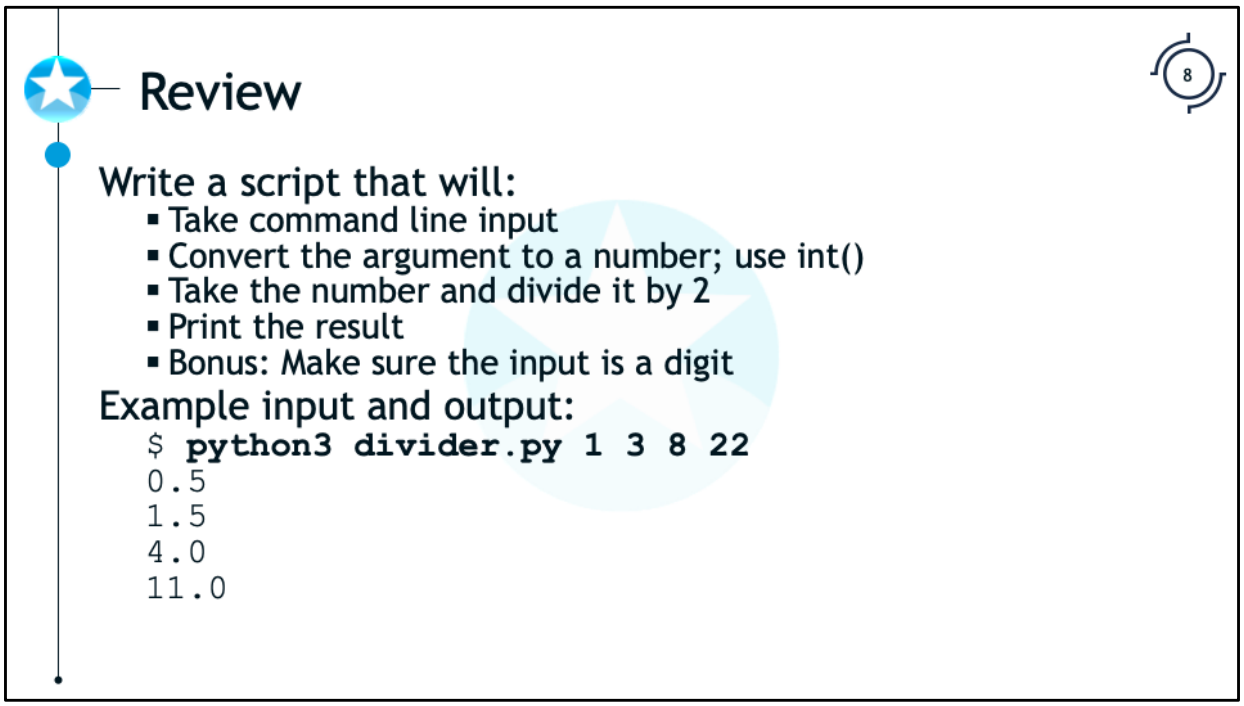
```
def myfunction:
    print('in my function')
```

Functions can also accept parameters by putting them in parenthesis.

```
def myfunction(a):
    print('in function with argument: ' + a)
```

You can have multiple arguments and separate each with a comma.

```
def myfunction(a, b):
    print('in my function')
    print('first argument: ' + a)
    print('second argument: ' + b)
```

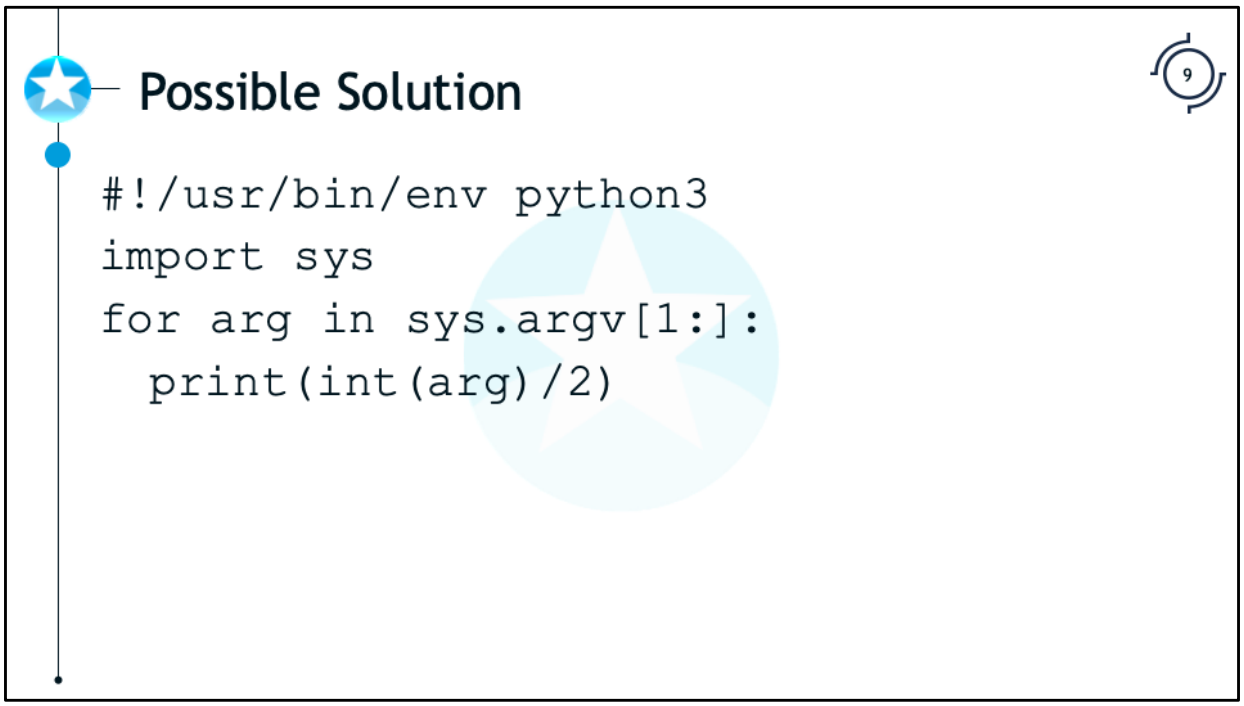


Write a script that will:

- Take command line input
- Convert the argument to a number; use int()
- Take the number and divide it by 2
- Print the result

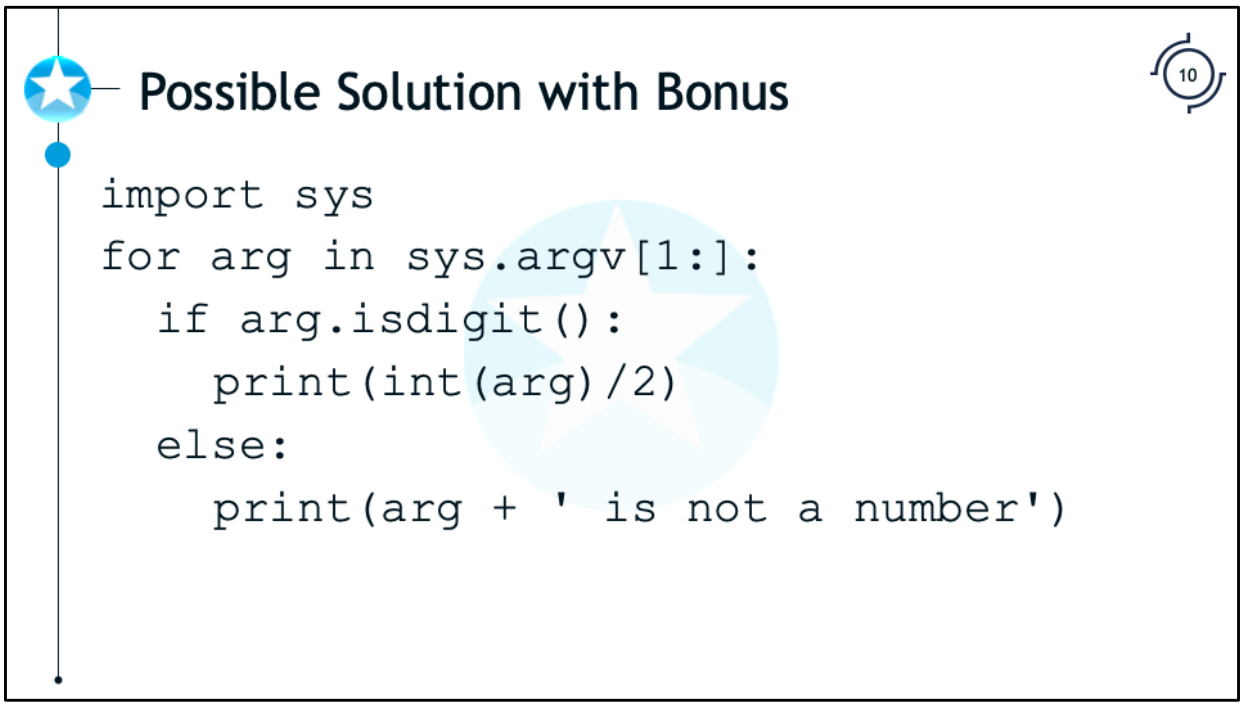
Example input and output:

```
$ python3 divider.py 1 3 8 22
0.5
1.5
4.0
11.0
```



A possible solution is shown below:

```
#!/usr/bin/env python3
import sys
for arg in sys.argv[1:]:
    print(int(arg)/2)
```

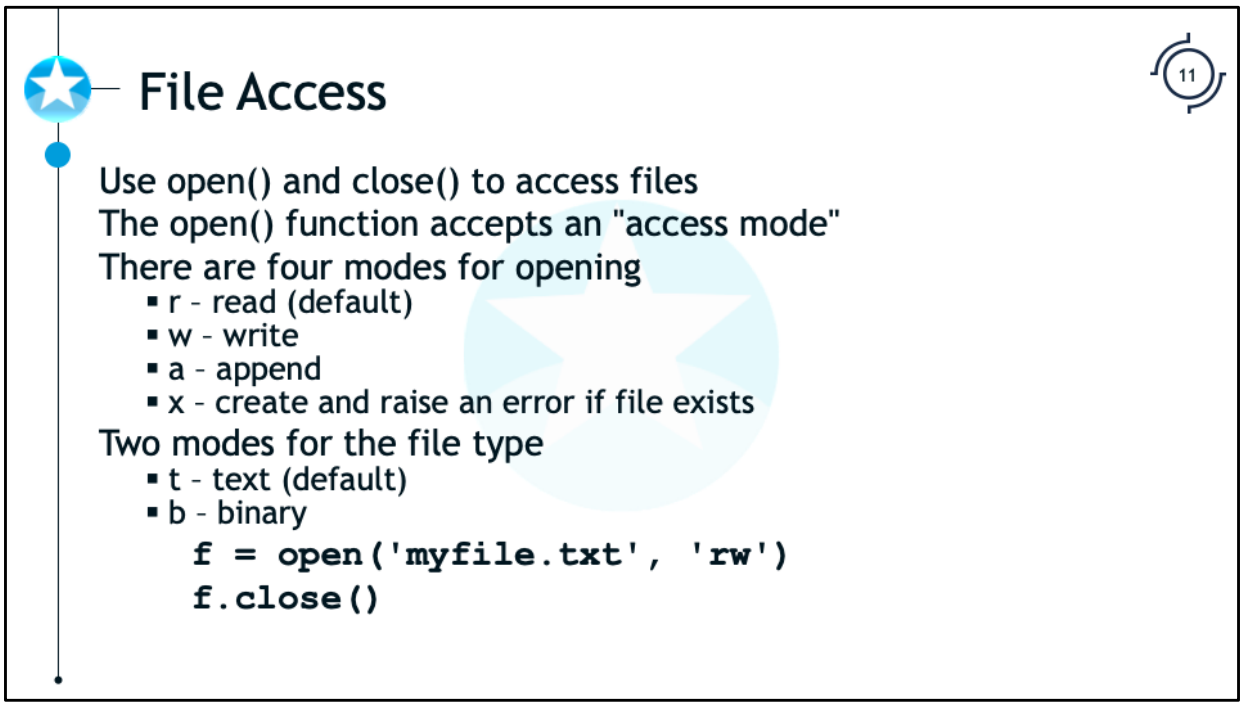


A possible solution is shown below:

```
import sys
for arg in sys.argv[1:]:
      if arg.isdigit():
            print(int(arg)/2)
      else:
            print(arg + ' is not a number')
```

If we give an invalid number (the letter "t") we will see this output:

```
$ python3 divider.py 1 3 8 22 t
0.5
1.5
4.0
11.0
t is not a number
```



Python can read and write files. The open() and close() functions will open and close a file handle, respectively. The open() function returns a file handle that we can use to access the file. When we are done with the file, we should call the close() method to release the file handle. When opening a file, there are four modes we can use (we can use more than one):

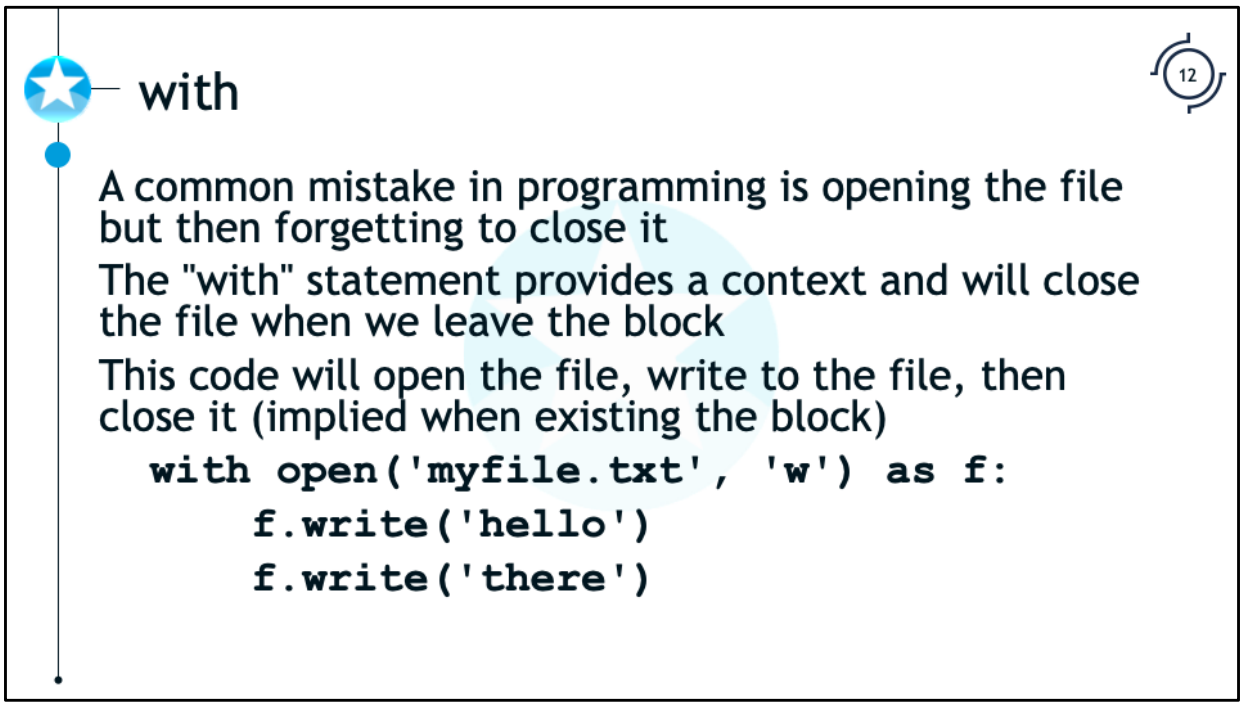
- r – read (default)
- w – write
- a – append
- x – create and raise an error if file exists

We can specify the contents of the file with:

- t – text (default)
- b – binary

The code below will open a file for read and write access in text mode.

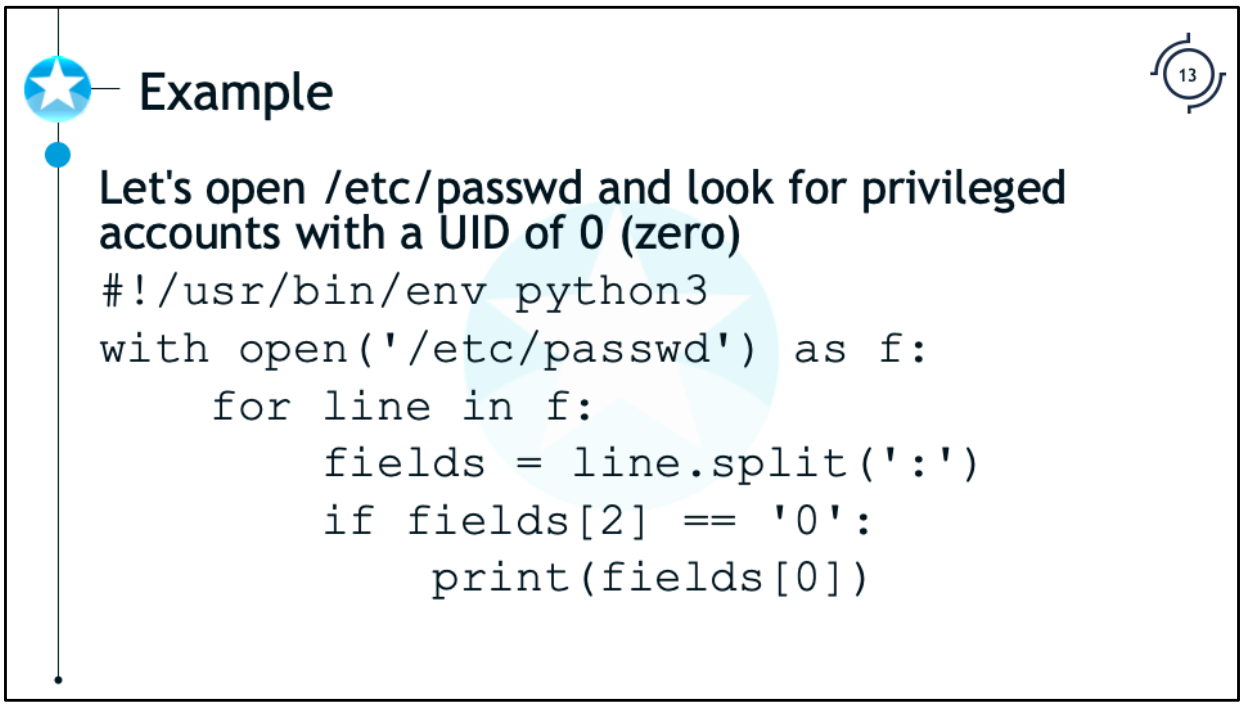
```
f = open('myfile.bin', 'rwb')
f.close()
```



A common mistake in programming is opening the file but then forgetting to close it. To simplify the closing we can use a the "with" statment. The "with" statement provides a context and will close the file when we leave the block. The code below will open the file, write to the file, then close it (implied when existing the block)

```
with open('myfile.txt', 'w') as f:
    f.write('hello')
    f.write('there')
```

The file handle "f" is only valid in the indented code under the "with" statement.

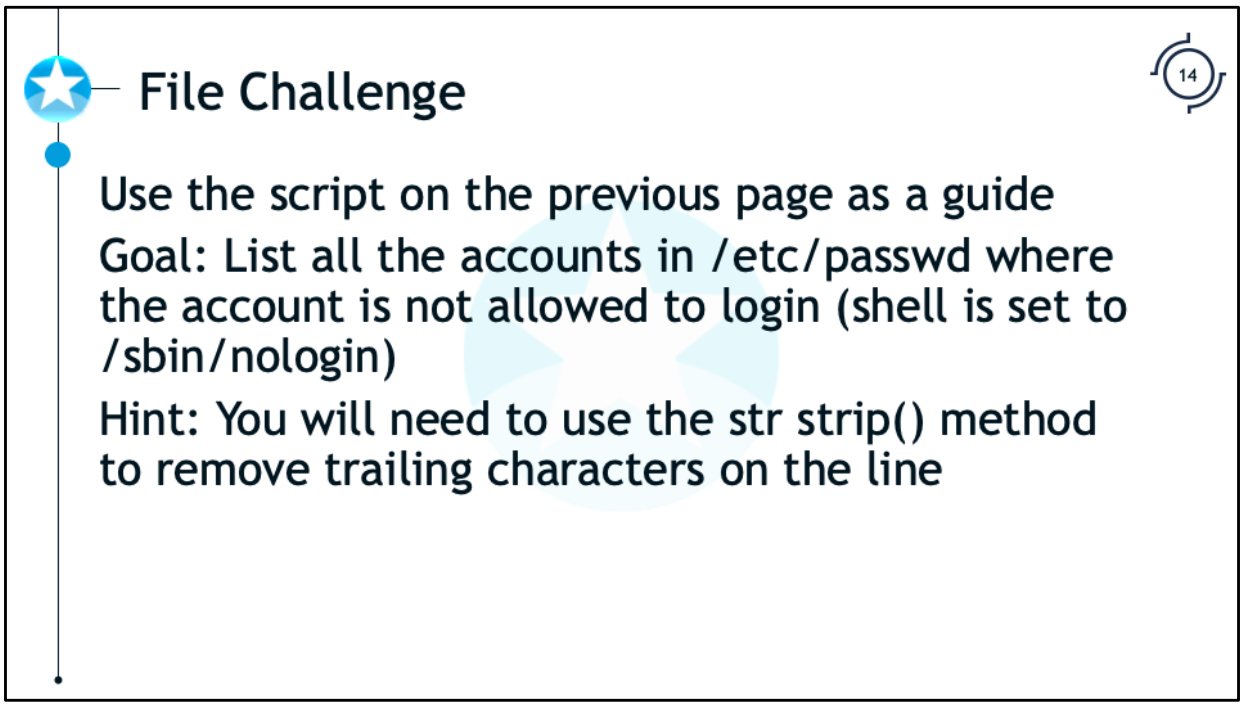


Let's open /etc/password and look for privileged accounts with a UID of 0 (zero). An example script looks like this:

```
#!/usr/bin/env python3
with open('/etc/passwd', 'r') as f:
    for line in f:
        fields = line.split(':')
        if fields[2] == '0':
            print(fields[0])
```

Let's discuss each line:

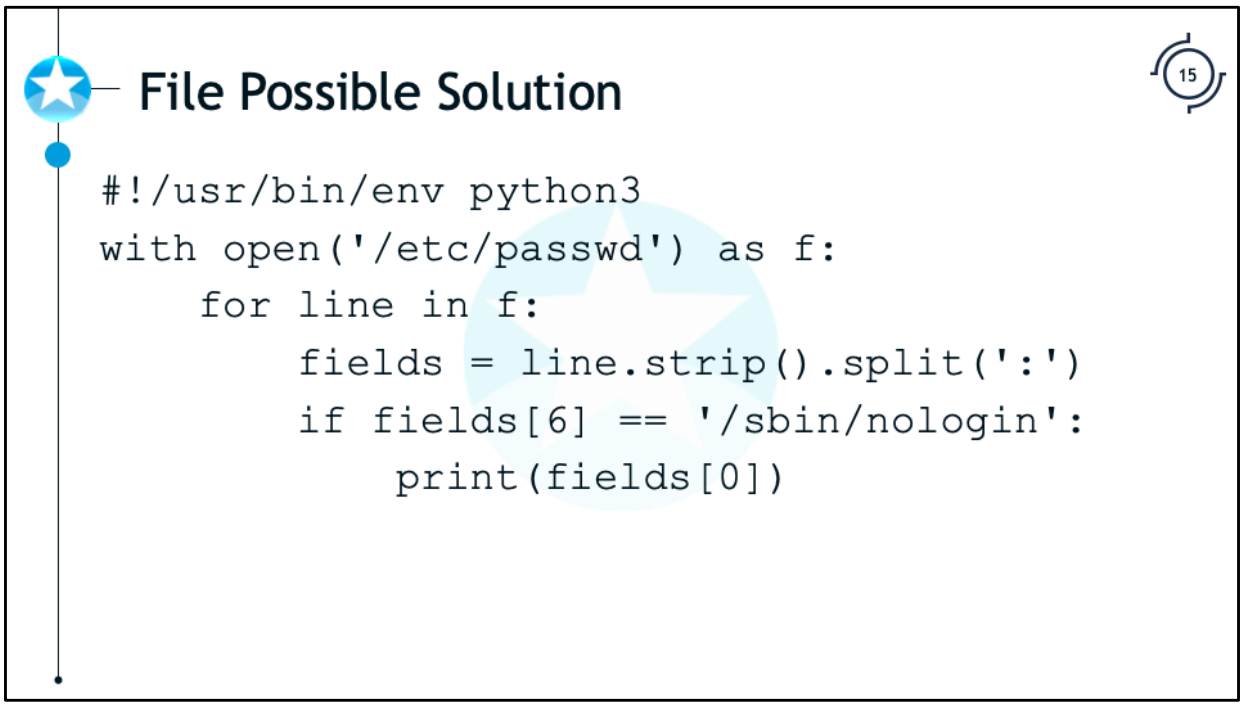
- The first line is the shebang line and specifies our interpreter
- The second line opens /etc/password in read-only mode and sets the variable "f" to the file object
- The for loop iterate through each line, one at a time where the variable "line" will present the line of text in /etc/passwd
- The fourth line splits the line using the colon as a delimiter (remember, that /etc/passwd and /etc/shadow are colon delimited fields
- The if statement checks the the third field (remember, the first item is index 0), the UID, is zero (root level)
- The next line prints the first (zeroth) field, the username



Use the script on the previous page as a guide. Your task is to list all the accounts in /etc/password where the account is not allowed to login (shell is set to /sbin/nologin).

Hint: You will need to use the str strip() method to remove trailing characters on the line.

Good luck!

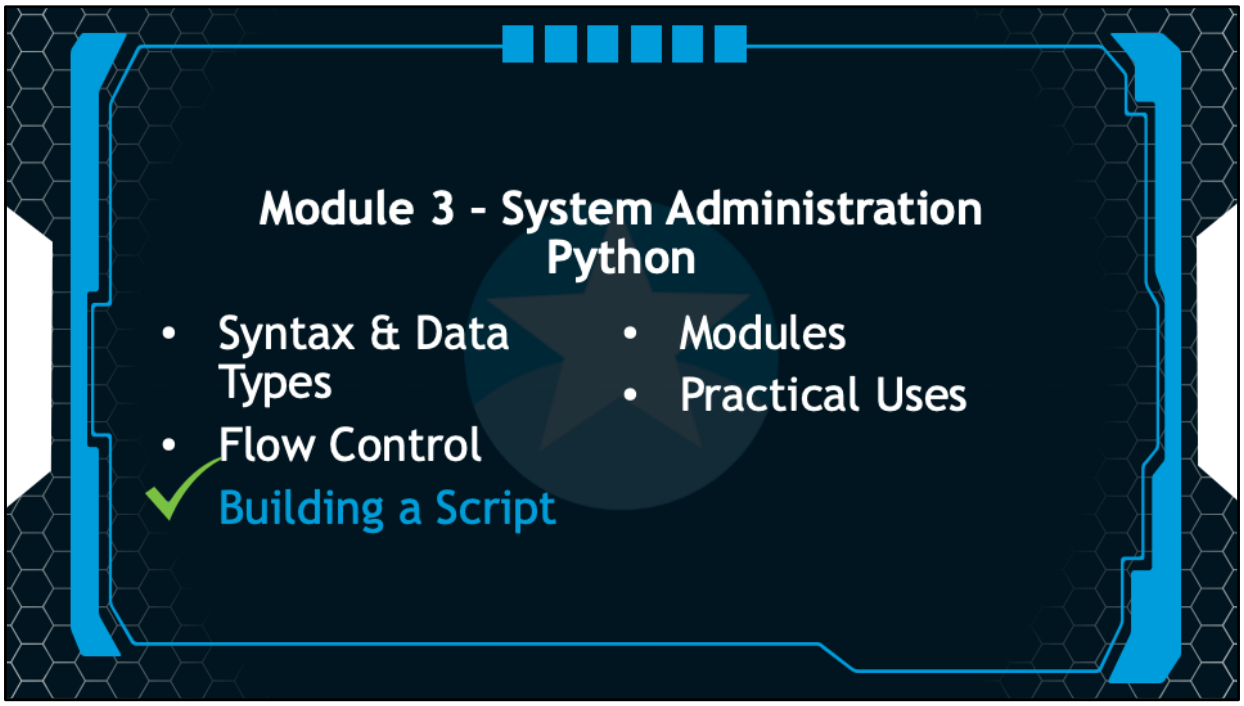


One possible solution is:

```
#!/usr/bin/env python3
with open('/etc/passwd', 'r') as f:
    for line in f:
        fields = line.strip().split(':')
        if fields[6] == '/sbin/nologin':
            print(fields[0])
```

The script is the 7th field (index 6). The problem is, the last field includes the newline character. We either need to add that to our search or to remove it. The strip() method in the str (string) object will remove this whitespace. We could remove the whitespace before or after the strip. The following two lines would have the same effect.

```
        fields = line.split(':')
        if fields[6].strip() == '/sbin/nologin':
```



In the next session we'll discuss using modules and introspection in Python.