Welcome to the Module 3, System Administration. In this sub-section we'll be discussing Python. In this session we'll discuss flow control in 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 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 will be introduced to flow control in Python.
The information here is the same as with Bash.

=  Assignment
+  Plus
−  Subtraction
*  Multiplication
/  Division
** Exponent
%  Mod (remainder)
+=  Plus-equal (increment)
−=  Minus-equal (decrement)
*=  Times-equal (self multiply)
/=  Slash-equal
%=  Mod-equal
The comparison operators are shown here:

```
==        Equals
!=        Not equals
<>        Not equals (similar to !=)
>         Greater than
<         Less than
>=        Greater than or equal to
<=        Less than or equal to
in        True if value is in list or tuple
not in    Inverse of in
and       Logical AND
or        Logical OR
not       Logical NOT
```
We can execute code based some condition using an "if" statement. We can do a simple statement like this:

```python
if x == 0:
    print('x is zero')
```

We can have two options with this code:

```python
if x == 0:
    print('x is zero')
else:
    print('unknown condition')
```

Or a more complex statement like this:

```python
if x == 0:
    print('x is zero')
elif x >= 1 and x < 10:
    print('x is a single digit')
elif x > 10 and x < 100:
    print('x is two digits')
else:
    print('unknown condition')
```

The indentation of each portion should be consistent. Some developers uses spaces (2, 3, 4, 5, 8) and some use tabs (just one) for each level of indentation. The choice of tabs and spaces (and further, the number of spaces) is a hotly contested topic in the development community. What really matters is consistency in a development project. Space People and Tab People both agree that if you mix tabs and spaces you
are a terrible person.
Loops

There are two types of loops

- **while** - repeats code while a condition is true
- **for** - executes code multiple times (counter)

**Special loop statements**

- **break** - exit the loop and continue the code after the loop
- **continue** - skip the remainder of the loop and go to the next iteration

We use loops to perform an action a number of times. There are two types of loops in Python, the while loop and the for loop.

The while loop will execute code while a condition is true. The for loop is used to loop a predetermined number of times, such as with a counter or over a list (or tuple).

Inside the loop there are special statements to control execution. The break statement is used to leave the entire loop. The continue statement is used to skip the current iteration, perform the test condition (if applicable) and then go to the next iteration.
We can use the for loop with a counter. The range function will give us a serious of numbers. The range statement can be used to give a simple range as shown above.

```python
>>> for i in range(3):
...     print(i)
...
0
1
2
```

Notice the prompt changes once you indent

```python
range(inclusive start, exclusive end, step)
```

We can also define the start, the stopping number, and step (increment) for each iteration. We could even use a negative number.

```python
>>> for i in range(4, 10, 2):
...     print(i)
...
4
6
8
```
We can also iterate over a list, tuple or other sequence of objects.

```python
>>> weekdays = ['m', 't', 'w', 't', 'f']
>>> for day in weekdays:
...     print(day)
...
mon
tuew
wed
thurs
fri
```
The while loop will run as long as the test condition is true. In this loop, we start with the number 1 and double it as long as it is less than 128.

```python
>>> n = 1
>>> while n < 128:
...     print(n)
...     n *= 2
... 
1
2
4
8
16
32
64
128
```
Most programs will take some sort of input. To accept keyboard input we use the `input()` function.

```python
x = input('Do you like Python? y/n ')
if x == 'y':
    print('Excellent!')
```

The code above will take input from the user and print "Excellent!" if the user enters "y" followed by the enter key.
Python doesn't have an unguarded loop since the while loop checks the condition on entry. To prevent multiple input and condition checks you will often see code like this:

```python
while True:
    x = input('Pick a number between 1 and 100: ')
    if x.isdigit() and int(x) >= 1 and int(x) <= 100:
        break
```

Without `break` we would see something like this:

```python
x = input('Pick a number between 1 and 100: ')
if not (x.isdigit() and int(x) >= 1 and int(x) <= 100):
    while not(x.isdigit() and int(x)>=1 and int(x) <=100):
        x = input('Pick a number between 1 and 100: ')
```

As you can see, there are multiple checks and multiple input requests which is more prone to mistakes.
The "continue" will skip the rest of the code in the current iteration and jump back to the top of the loop. If it were a while loop the condition would be checked before execution. The code below demonstrates the usage of "continue".

```python
>>> for letter in 'python':
...     if letter == 'h':
...         continue
...     print(letter)
...
python
```
Using what you've learned here so far, write Python that does the following:
- Asks the user if they like Python
- Only accepts "y" or "n" as input
- If "y", then print "Excellent"
- If "n", then print "Sorry"
- If any other entry, print "Invalid input"
One possible answer is shown here:

```python
while True:
    a = input('Do you like Python? y/n ')
    if a == 'y':
        print('Excellent')
        break
    elif a == 'n':
        print('Sorry')
        break
    else:
        print('Invalid input')
```
We've completed the discussion of flow control in Python. In the next session, we'll discuss reusing code in scripts and functions.