List Comprehensions for Creating Lists

Advanced Topic

- Shortcut for creating lists; useful but can be a little confusing
- Simplest syntax: `[ expression for value in list ]`

```
dev_info = '  1.1.1.1, username, password   '
list = [item.strip() for item in dev_info.split(',')]
['1.1.1.1','username','password']
```

- Advanced: `[ expression for value in list if condition ]`

List comprehension – when you see [ ]

- Will apply a for loop and apply some expression
- To interpret start at the for loop and then understand the expression

List comprehensions are a short way of creating a list, using syntax that is not regularly found in other languages.

Lists are typically created using normal `for` loops or nested `for` loops. Consider an unstructured set of data consisting of a device IP address, username, and a password:

```
device_string = '  1.1.1.1, username  , password   '
```

To create a list that does not include the extra white space above, you could do the following:

```
info_list = list()  # create empty list
device_info = device_string.split(,)  # create list from device_string
for item in device_info:
    item = item.strip()  # strip white space from device
    info_list.append(item)  #add to info_list
```

List comprehensions do these two things at once: the `for` statement and the assignment of an item in the list. The trick in identifying a list comprehension is that you see square brackets enclosing what appears to be a `for` statement:

```
[expression for value in list]
```

To accomplish the same task using a list comprehension:

```
device_string =  '  1.1.1.1, username  , password   '
info_list = [ device.strip() for device in device_string.split(',') ]
```

Looking at the right-hand side of the statement, the square brackets indicate that this code must be a comprehension of some type. Inside the bracket is the statement:

```
for item in device_string.split(',')
```

This `for` statement will create a list using the split function

```
device_string.split(',')
```

This string manipulation function will take the original string (device_string), and create a list using the comma (',') as the separator for the list. The list created is:

```
['   1.1.1.1', ' username  ', ' password    ']
```

The 'split' functionality provides the individual items in the string – IP address, username, password – but a lot of white space remains. It is against this list that the `for` statement operates. So the `for` statement is really:

```
for item in ['   1.1.1.1', ' username', ' password    ']
```

For every iteration, the value of 'item' will be an item from that list. So the first time through, item will equal:

```
item = '   1.1.1.1'
```

The list comprehension, for the first iteration, looks as follows:

```
[ item.strip() ]
```

The `strip` string function strips off white space at the beginning and end of the string. So now, for the first iteration, the list comprehension value will be:

```
'1.1.1.1'
```

For the second and third iterations of the `for` loop, the values will be:

```
'username'
'password'
```

The list that is created by the list comprehension looks like:

```
['1.1.1.1', 'username', 'password']
```

This list comprehension was complicated but hopefully one that would be of use to you as a network engineer. Examples of list comprehension abound outside of this training, if you have further interest.