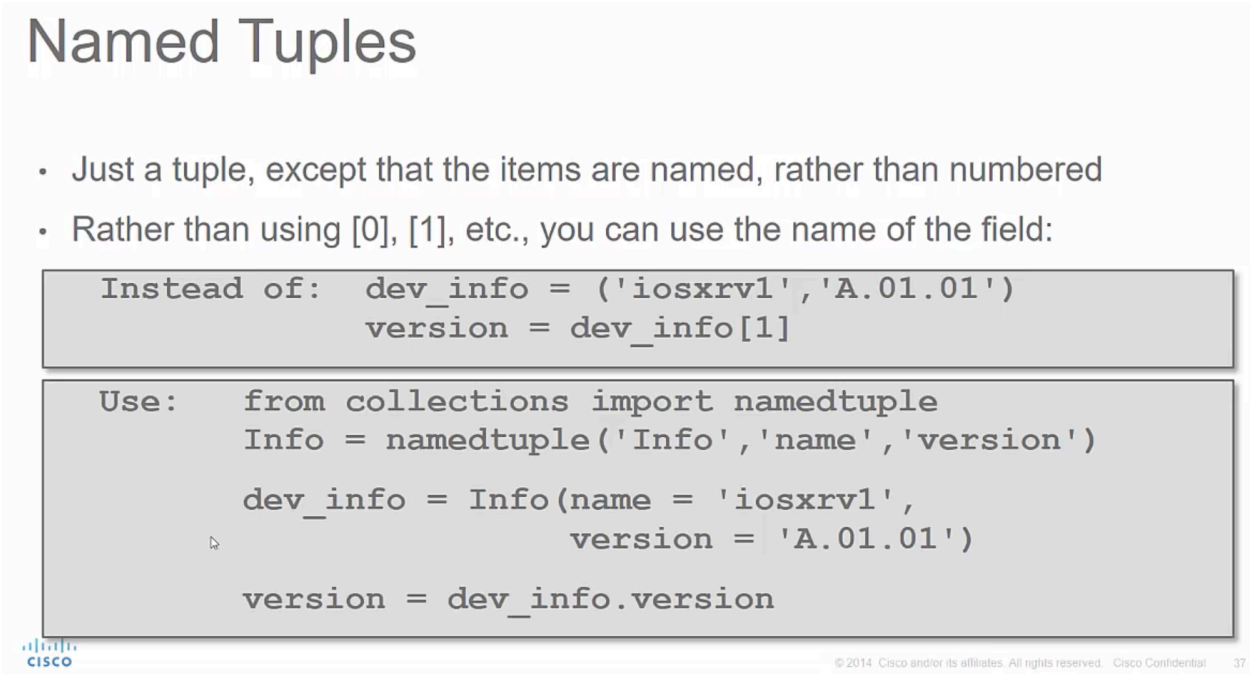

Named tuples can provide significant improvements in code readability. Tuples suffer from the fact that in certain cases, the items that are stored are not a collection of the same type of item, but are actually different types of items.

In such cases, it would be helpful to be able to refer to specific items in the tuple by name, rather than by index. For example, with the device information tuple, it would be nice to refer the IP address by name, the username by name, and the password by name, rather than by index.

The following example shows how named tuples could be used to improve code readability. Consider the indexed way of doing things. For the dev_info tuple, the version field is accessed by using the numerical index of its location in the tuple, which is 1:

```
dev_info = ('iosxrv1', 'A.01.01')
version = dev_info[1]
```

Contrast the example with using a named tuple. The setup takes some extra code, but referencing the field within dev_info is much cleaner:

```
from collections import namedtuple
Info = namedtuple('Info','name','version')
dev_info = Info(name='iosxrv1', version='A.01.01')
version = dev_info.version
```

To use a named tuple, you first declare that you are using named tuple by importing it. Then you create an object class for your device information, in which you define the fields (name and versions).

Creating the tuple itself involves declaring your named tuple class 'Info' and specifying the values for the named fields, 'name' and 'version', which are set to 'iosxrv1' and 'A.01.01' respectively.

At this point, your tuple is specified using the actual name for the field, which is the version.

Named Tuples are a popular data structure in python

```
from collections import namedtuple
from pprint import pprint

Dev_info = namedtuple('Dev_info',['name', 'os_type', 'ip', 'user', 'password'])

os_types = set()

file = open('devices','r')
for line in file:

    device_info = Dev_info(*(line.strip().split(',')))
    print 'Device Information: ', device_info

    if device_info.os_type not in os_types:
        os_types.add(device_info.os_type)

pprint(os_types)
```

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section07$ python S03-10-set-os-type.py
Device Information:  Dev_info(name='device1', os_type='ios', ip='10.3.21.5', user='user1', password='pass1')
Device Information:  Dev_info(name='device2', os_type='ios', ip='10.3.21.6', user='user2', password='pass2')
Device Information:  Dev_info(name='device3', os_type='nx-os', ip='10.3.21.7', user='user3', password='pass3')
Device Information:  Dev_info(name='device4', os_type='nx-os', ip='10.3.21.8', user='user4', password='pass4')
Device Information:  Dev_info(name='device5', os_type='ios-xr', ip='10.3.21.9', user='user5', password='pass5')
Device Information:  Dev_info(name='device6', os_type='ios-xr', ip='10.3.21.10', user='user6', password='pass6')
set(['ios', 'ios-xr', 'nx-os'])
```