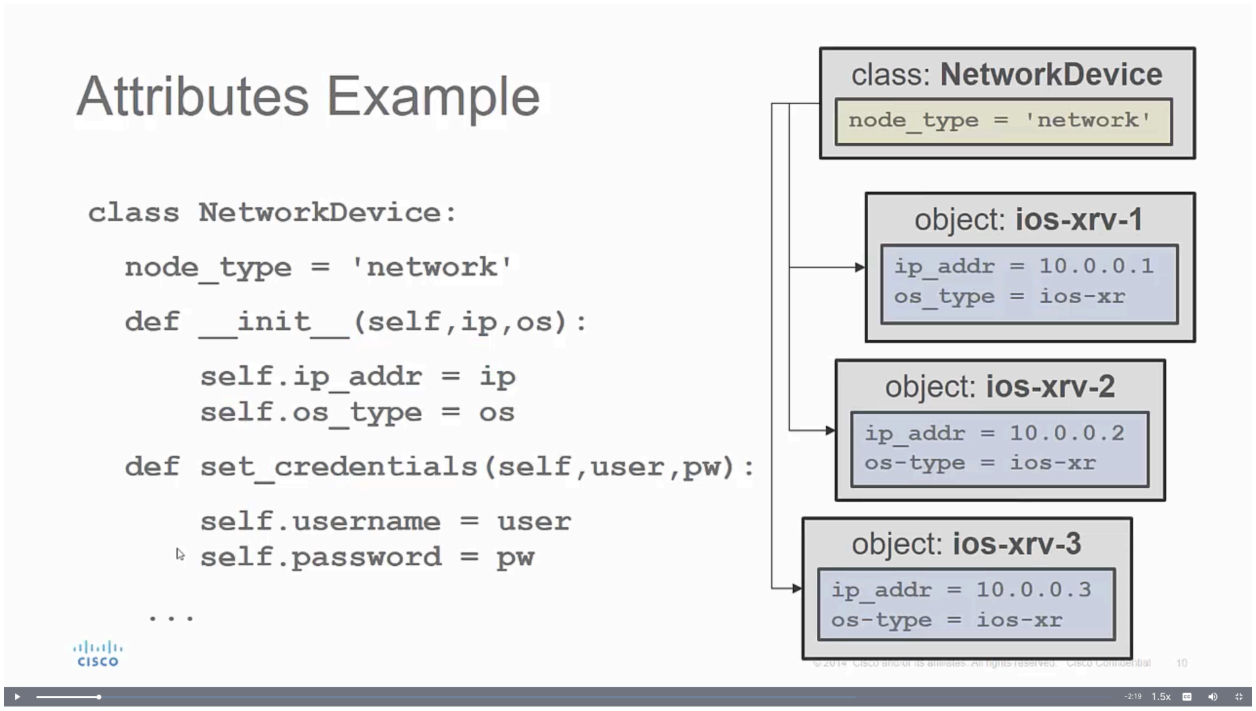
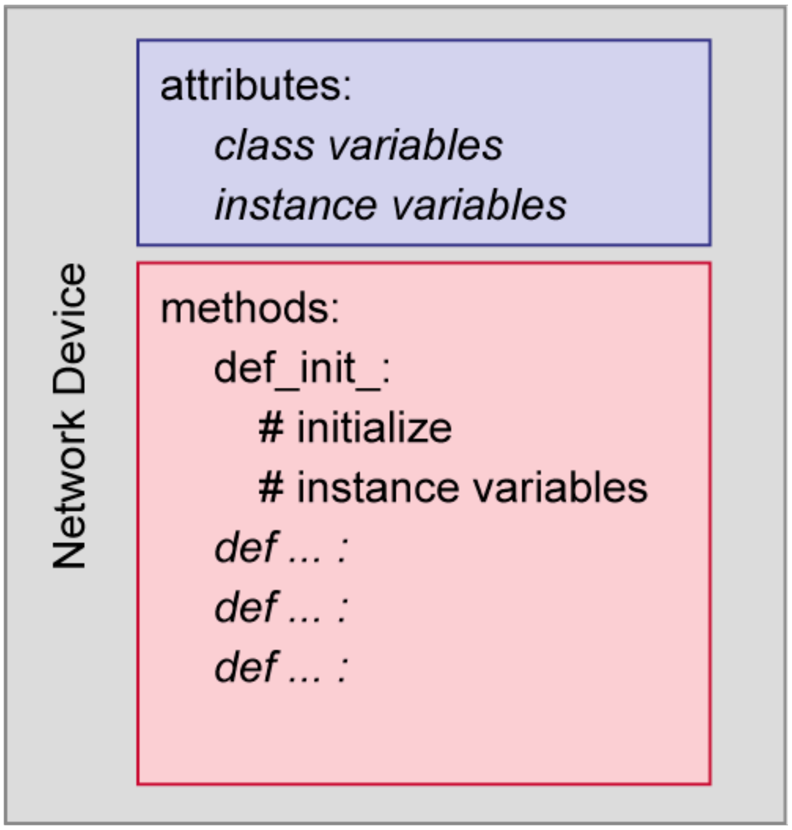


Attributes Example

```
class NetworkDevice:
  node_type = 'network'
  def __init__(self,ip,os):
      self.ip_addr = ip
      self.os_type = os
  def set_credentials(self,user,pw):
      self.username = user
      self.password = pw
    ...
```

"Self" will precede attributes that will be used when the object is instantiated

The attributes for a class can be of any type–string, number, list, dictionary, complex combinations of data structures, and so on. They can even be other classes; for example, if you have a NetworkDevice class, you may also have an Interface class, and your NetworkDevice would have a list of Interface objects.


Network Device
attributes:
class variables
instance variables
methods:
def_init_:
# initialize
# instance variables
def ... :
def ... :
def ... :

There are a few things to be aware of regarding object-oriented attributes in Python:

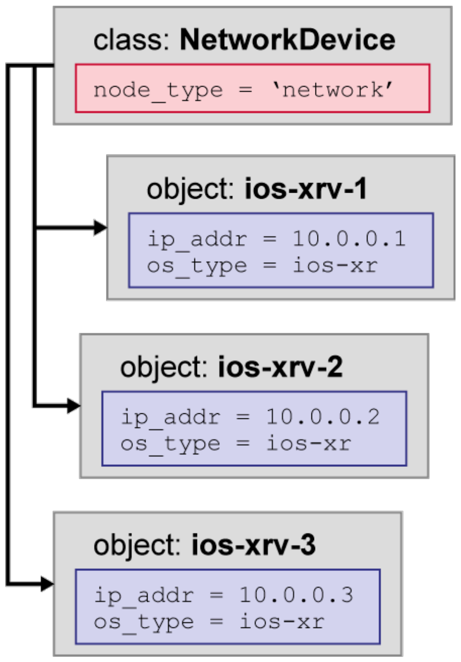
- **Created when used:** Languages with static typing require the programmer to explicitly define all attributes of a class. Because Python does not require declaration of variables, it likewise does not require the declaration of attributes. Instead, just like in the rest of the language, Python creates attributes only when they are first used. If the only time you use an attribute is in a specific method call, that attribute will not exist until that function is called. Therefore, it is a good practice to initialize all your attributes during the initialization of your object. That initialization is done in the '__init__' call, described next.
- `init()`**:** When object instances are initialized, the __init__ function for your object is called. The __init__ function is where you have a chance to set up your object and is the place to initialize your instance attributes to their appropriate value.
- **'self':** Python requires that the first parameter for every method be 'self'. This 'self' parameter is a reference to this specific object instance.

When you reference an instance attribute, you must precede the attribute name with the name of your object instance, which is 'self'.

The example which follows helps to describe these facets of Python attributes for classes and objects.

Attributes Example

The example shows how a 'NetworkDevice' class might be constructed, showing both attributes and methods.



The code for this class definition is as follows:

```
class NetworkDevice:
   node_type = 'network'
   def __init__(self,ip,os)
      self.ip_addr = ip
      self.os_type = os
   def set_credentials(self,user,pw)
      self.username = user
      self.password = pw
```

Things to note:

```
class NetworkDevice:
```

The keyword `class` identifies what follows as a class definition. The name that follows is the name of the class. By convention, class names use what is commonly called 'camel-case', meaning that the first letter of each word in the name is capitalized, without the typical underscores ('_').

```
def __init__(self,ip,os):
def set_credentials(self,user,pw):
```

The keyword `def` within a class definition is used to identify the methods of the class. In the first case, the method is the initialization method, which must be named '__init__'. Notice that for this method, and for the 'set_credentials' method, the first parameter is always 'self', which is then used within the code to reference instance attributes.

```
self.ip_addr = ip
self.os_type = os
```

Code within each method will access instance attributes but prepending the attribute name with 'self.', for example, `self.attribute,` which tells Python that you are referencing an attribute which is unique to this specific object.

```
node_type = 'network'
```

One last item – the example placed a class attribute into the class, to show the difference between class and instance attributes. You will use instance attributes far more often than class attributes, but it is important to remember that instance attributes must always have 'self.' preceding them.