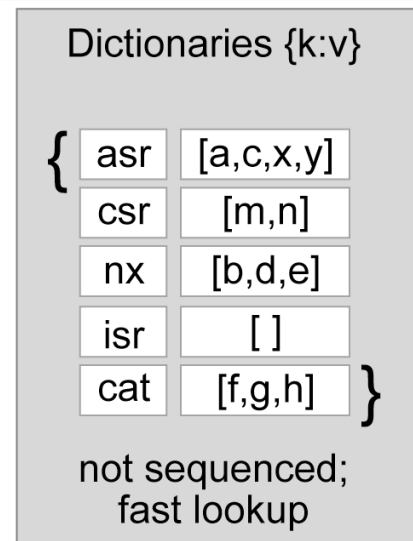
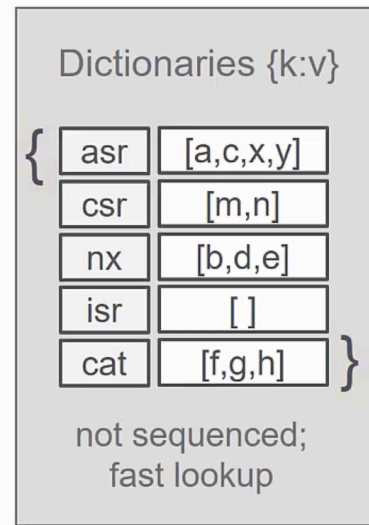


Overview of Dictionaries {...}

- **Unordered:** Not sequenced, accessed by key
- **Key:** Key must be hashable (string or numeric even tuple, but not list, dictionary, or set)
- **Mutable:** Items can be added, removed, changed
- **Items:** Item values can be simple or complex (e.g. lists, tuples, or dictionaries)



- **Unordered:** Not sequenced, accessed by key
- **Keys:** Key must be hashable—string, numeric, or tuple
- **Mutable:** Items can be added, removed, or changed
- **Items:** Items can be simple or complex.

Creating a Dictionary

Create empty dictionary

- Using the dict() function

```
dev1 = dict() # creates empty dictionary "dev1"
```

- Using empty curly braces { }

```
dev1 = {} # creates empty dictionary "dev1"
```

Create populated dictionary

```
dev1 = {"ip": "10.3.21.5", "version": "A.01.01",  
        "username": "user1", "password": "pw1"}
```

	key	value
dev1	'ip'	'10.3.21.5'
	'version'	'a.01.01'
	'username'	'user1'
	'password'	'ps1'

Dictionaries are another extremely useful data structure in Python. A few attributes of dictionaries:

- **Unordered:** Dictionaries are not sequenced; hence they are 'unordered', which means that you don't typically iterate through a dictionary. You go specifically to an individual item using the 'key' value.
- **Keyed:** For every value in a dictionary, there is a key by which it is referenced. These keys must be 'hashable', meaning they must be capable of going through a process to create a 'hash' value for lookup purposes. Strings and numbers are hashable; and because tuples are immutable, they too can be used as keys in a dictionary. Lists, dictionaries, and sets cannot be used as keys.
- **Mutable:** Items in a dictionary can change, hence there are facilities for adding and removing items in a dictionary.
- **Items:** The values that are stored in a dictionary can be simple values or they can be data structures such as lists, other dictionaries, tuples, or sets.

Some characteristics of dictionaries include:

- Python uses the curly braces '{}' to enclose dictionaries.
- Keys can be hashed, as in the figure where Cisco device types are the dictionary keys.
- Values can be single values or they can themselves be data structures – in the figure there are lists for each dictionary item. The intention in the abbreviated example is that the dictionary holds keys for each device type, and the actual value for each key is a list of devices. In the example, the device details are shown as simple letters, but in a real situation they would be IP addresses or the hostnames of the devices.

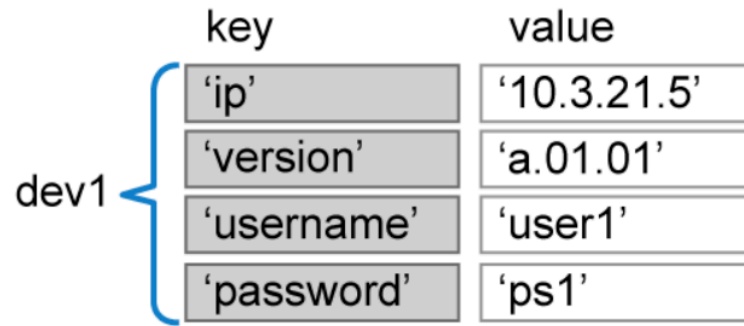
Creating a Dictionary

As with lists, there are two methods for creating an empty dictionary. The creation of a populated dictionary involves specifying the keys and values for each item separated by a colon (':').

- Create an empty dictionary

```
dev1 = dict () # method 1  
  
dev1 = {} # method 2
```

- dev1 = {"ip":"10.3.21.5", "version":"A.01.01", "username":"user1", "password":"pw1"}



	key	value
dev1 {	'ip'	'10.3.21.5'
	'version'	'a.01.01'
	'username'	'user1'
	'password'	'ps1'

Creating an Empty Dictionary

```
dev1 = dict() # creates empty dictionary "dev1"  
dev1 = {}     # creates empty dictionary "dev1"
```

The first example above creates the empty dictionary using the `dict` function. The second example does the same thing, but uses the empty curly braces method.

Creating a Populated Dictionary

```
dev1 = {"ip":"10.3.21.5", "version":"A.01.01", "username":"user1","password":"pw1"}
```

The example above is different from the dictionary example at the beginning of this lesson. The previous example had Cisco device types as the keys, and the value for each item was a list. In that case, all items were homogeneous, meaning they were of the same type.

In the example above, however, the items are technically all the same 'type', in that they are all strings. However, each item is holding something different – IP address, firmware version, username, password. In more advanced circumstances, this dictionary would probably not be using a string for IP address, but a class holding the actual binary value. The point however is to understand that dictionaries can be used to hold heterogeneous types of data.

You can see in the example that the data is defined using what are called 'key-value' pairs. The keys and values are separated by a colon, with the keys on the left, and the value on the right. In the following piece of the example:

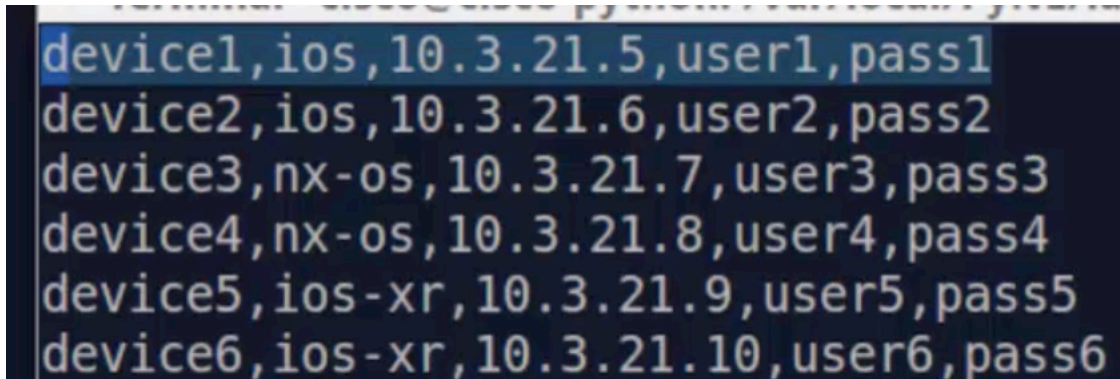
```
{"ip":"10.3.21.5", "version":"A.01.01", "username":"user1","password":"pw1"}
```

The curly braces begin and end the dictionary. The first object that is specified is the key ("ip"); the second object that is specified is the value ("10.3.21.5").

Items in the dictionary are separated by commas.

You will encounter dictionaries that are defined in this way quite often in Python.

You may also notice that the format for defining a dictionary is quite similar to the encoding used with JSON. You will find that for doing many networking tasks, especially for an SDN you will end up using both Python dictionaries and JSON-formatted data.



```
device1, ios, 10.3.21.5, user1, pass1
device2, ios, 10.3.21.6, user2, pass2
device3, nx-os, 10.3.21.7, user3, pass3
device4, nx-os, 10.3.21.8, user4, pass4
device5, ios-xr, 10.3.21.9, user5, pass5
device6, ios-xr, 10.3.21.10, user6, pass6
```

```
from pprint import pprint

device_info = {} # Create my device_info dictionary

# Open the file and read in the single line
file = open('devices','r')
file_line = file.readline().strip()

print 'read line: ', file_line # Print out the line I just read

# Here is the main part: use the string 'split' to create a list
# of items that are separated by commas
device_info_list = file_line.split(',')

# Now put those items from the list into our dictionary
device_info['name'] = device_info_list[0]
device_info['os-type'] = device_info_list[1]
device_info['ip'] = device_info_list[2]
device_info['username'] = device_info_list[3]
device_info['password'] = device_info_list[4]

pprint(device_info) # Print out the dictionary with nice formatting
file.close() # Be a good steward of resources and close the file
```