Introduction to Python Storage

- Human-readable
- Simple persistence
- Iterable
- Testing

- Human-readable
- `JSON, CSV, XML`
- Conversion to Python data formats

- Not human-readable
- Efficient storage
- Binary data such as keys, images

- Not human-readable
- Large volumes of data
- Fast search capabilities

Reading and Writing Files

Text files:

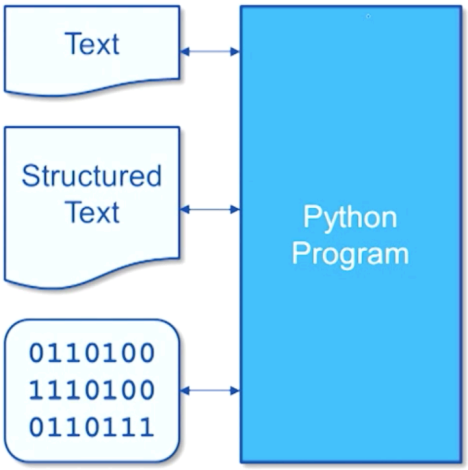
- `read(), readline(), write()`

Structured text files:

- JSON: `import json`
- CSV: `import csv`
- XML: `import xml.dom, xml.sax`

Binary files

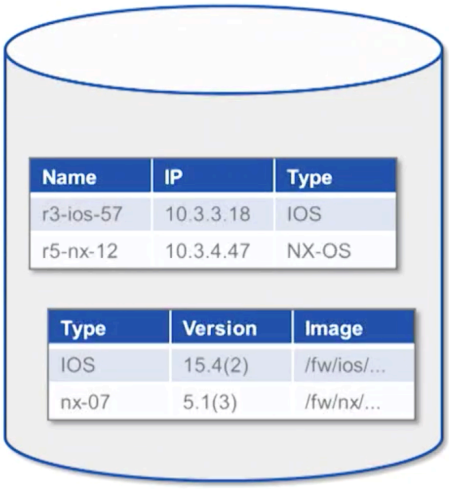
- `open(filename, 'rb')`



Database Overview

Reasons for using a database

- **Volume:** Large volumes of data that would exceed capacity for normal files
- **Tables:** Multiple tables of information, e.g. devices, users, locations, policies
- **Search:** Searching for a specific item of data
- **Filter:** Retrieving all data that meet a certain criteria



Once your network programmability application matures beyond the initial testing and prototype state, you will likely want to utilize some type of storage mechanism beyond an unstructured text file. This section discusses various mechanisms provided by Python for storing your networking data.

In general there are four categories of data storage in Python:

- **Unstructured Text:** A basic text file following no specific format. Easy to iterate in a `for` loop. These types of files can be used for prototyping and testing your code. When you are building 'production' code, you will probably want to use a structured text file of some type.
- **Structured Text:** There are tools that are provided in Python for dealing with structured files as well. Popular formats include comma-separated-value (CSV), JSON, and XML. Libraries exist for reading and writing these files, and converting them to and from typical Python data structures.
    - **CSV:** Compact table-like data with items separated by commas, each line in the file being like a row in a table or spreadsheet. Use these files for storing lists of homogenous data, such as a list of devices, log files, or something such as time series data.
    - **JSON:** Simple human-readable format for storing arrays (like Python lists) and objects (like Python dictionaries). These types of files are useful for storing dictionary-type data, with names (keys) and values. Used for storing data structures which may vary, requiring names for each of the data items. Device information which may vary from device type to device type would be a good candidate for a JSON file. JSON however is very popular and efficient, and so can be used for other types of data such as logs, time-series data, and just about any other type of data.
    - **XML:** Highly structured, less human-readable than JSON. XML can benefit from a data definition file which helps translate its data. XML was once popular but in many cases, XML has been replaced by JSON.
- **Binary:** Your application may need to deal with binary data as well, for things such as software images or certificates. Python provides mechanisms for dealing with these types of files as well. Binary files can be useful for storing binary data (software images, certificates), and also for storing general data more efficiently, such as time-series data representing numeric values. In such cases, rather than storing the information in a structured textual file, storing it in binary may be a faster and more efficient option.
- **Database:** SQL databases for storing tables, large volumes of data, and for performing complex searches. At some point, you may desire the use of a real database for storing large amounts of data, for searching, and for normal relational database operations such as maintaining tables of data and relationships between items in those tables. Python libraries exist for performing create, read, update, and delete SQL database operations on such data.

Python with Statement

When dealing with files, there is a fair amount of overhead that is involved with opening and closing a file. The `with` statement simplifies file operations.

Normally you would perform an `open` to open the file, and a `close` to close it when finished:

```
file = open('myfile','r')
    ... # do something with the file
file.close()
```

Using the `with` statement:

```
with open('myfile','r') as file:
```

The `with` statement will take care of closing the file for you when you exit the code block. In the examples of these lessons, you will see the Python `with` statement in action.

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section20/S10-2-json$ cat json-devices
[
  {"name":"ios-01","os":"ios","ip":"10.30.30.1","user":"cisco","password":"cisco"},
  {"name":"ios-02","os":"ios","ip":"10.30.30.2","user":"cisco","password":"cisco"},
  {"name":"ios-03","os":"ios","ip":"10.30.30.3","user":"cisco","password":"cisco"}
]
```

```
import json
from pprint import pprint

from devclass import NetworkDevice
from devclass import NetworkDeviceIOS
from devclass import NetworkDeviceXR

#=====================================================================
def read_devices_info(devices_file):

    devices_list = []

    # Open the device file with JSON data and read into string
    json_file = open(devices_file,'r')   # open the JSON file
    json_device_data = json_file.read()  # read in the JSON data from file

    # Convert JSON string into Python data structure
    devices_info_list = json.loads(json_device_data)

    for device_info in devices_info_list:

        # Create a device object with this data
        if device_info['os'] == 'ios':
```

```
            device = NetworkDeviceIOS(device_info['name'],device_info['ip'],
                                      device_info['user'],device_info['password'])

        elif device_info['os'] == 'ios-xr':

            device = NetworkDeviceXR(device_info['name'],device_info['ip'],
                                     device_info['user'],device_info['password'])

        else:
            device = NetworkDevice(device_info['name'],device_info['ip'],
                                   device_info['user'],device_info['password'])

        devices_list.append(device) # Append this device object to list

    return devices_list
#=============================================================================
def print_device_info(device):

    print '------------------------------------------------------------'
    print '    Device Name:       ',device.name
    print '    Device IP:         ',device.ip_address
    print '    Device username:   ',device.username,
    print '    Device password:   ',device.password
    print '------------------------------------------------------------'
#=============================================================
def write_devices_info(devices_file, devices_list):

    print '---- Printing JSON output -----------------------------'

    # Create the list of lists with devices and device info
    devices_out_list = []  # create list for JSON output

    for device in devices_list:
        dev_info = {'name':device.name,'ip':device.ip_address,'os':device.os_type,
                    'user':device.username,'password':device.password}
        devices_out_list.append(dev_info)

    pprint(devices_out_list)

    # Convert the python device data into JSON for output to file
    json_device_data = json.dumps(devices_out_list)

    # Output the JSON string to a file
    with open(devices_file, 'w') as json_file:
        json_file.write(json_device_data)
```

```
==== Device ===================================================================
--- connecting IOS: telnet 10.30.30.3
-------------------------------------------------------
    Device Name:       ios-03
    Device IP:         10.30.30.3
    Device username:   cisco     Device password:   cisco
-------------------------------------------------------
---- Printing JSON output -------------------------------
[{'ip': u'10.30.30.1',
  'name': u'ios-01',
  'os': 'ios',
  'password': u'cisco',
  'user': u'cisco'},
 {'ip': u'10.30.30.2',
  'name': u'ios-02',
  'os': 'ios',
  'password': u'cisco',
  'user': u'cisco'},
 {'ip': u'10.30.30.3',
  'name': u'ios-03',
  'os': 'ios',
  'password': u'cisco',
  'user': u'cisco'}]
```

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section20/S10-2-json$ cat json-devices-out
[{"ip": "10.30.30.1", "password": "cisco", "os": "ios", "name": "ios-01", "user": "cisco"}, {"
ip": "10.30.30.2", "password": "cisco", "os": "ios", "name": "ios-02", "user": "cisco"}, {"ip"
: "10.30.30.3", "password": "cisco", "os": "ios", "name": "ios-03", "user": "cisco"}]cisco@cis
```