Lab - Parse Different Data Types with Python

Objectives

Part 1: Launch the DEVASC VM

Part 2: Parse XML in Python

Part 3: Parse JSON in Python

Part 4: Parse YAML in Python

Background / Scenario

Parsing means analyzing a message, breaking it into its component parts, and understanding the purpose of each part in context. When messages are transmitted between computers, they travel as a stream of characters. Those characters are effectively a string. That message needs to be parsed into a semantically-equivalent data-structure containing data of recognized types (e.g., integers, floats, strings, and Booleans) before the data can be interpreted and acted upon.

In this lab, you will use Python to parse each data format in turn: XML, JSON, and YAML. We'll walk through code examples and talk about how each parser works.

Required Resources

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine

Instructions

Part 1: Launch the DEVASC VM

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

Part 2: Parse XML in Python

Because of the flexibility provided by Extensible Markup Language (XML), it can be tricky to parse. XML's all-text tagged data fields do not map unambiguously to default data types in Python or other popular languages. In addition, it is not always obvious how attribute values should be represented in data.

These issues can be sidestepped by Cisco developers working in some contexts, because Cisco has provided tools such as YANG-CLI, which validates and consumes XML relevant to data modeling and related tasks. Below is content of the **myfile.xml** file found in **~/labs/devnet-src/parsing**. This is an example of the sort of file that YANG-CLI manages. You will parse this file in Python to get access to the information it contains.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="1"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
   <candidate/>
```

```
  </target>
  <default-operation>merge</default-operation>
  <test-option>set</test-option>
  <config>
   <int8.1
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
    nc:operation="create"
    xmlns="http://netconfcentral.org/ns/test">9</int8.1>
  </config>
 </edit-config>
</rpc>
```

Step 1: Build a script to parse the XML data.

a. Open the **parsexml.py** file found in the **~/labs/devnet-src/parsing** directory.

b. Import the ElementTree module of the xml library and the regular expression engine. The ElementTree module will be used to do the parsing. The regular expression engine will be used to search for specific data.

Note: If you do not have any experience with using regular expressions in Linux, Python, or other object-oriented programming languages, search the internet for tutorials.

```
import xml.etree.ElementTree as ET
import re
```

c. Next, use the **parse** function from ET (ElementTree) to parse the **myfile.xml** file and assign it to a variable (**xml**). Then, get the root element with the **getroot** function and assign it to a variable (**root**).

```
xml = ET.parse("myfile.xml")
root = xml.getroot()
```

d. Now the top level of the tree can be searched for the containing tag **<edit-config>**, and when found, that tagged block can be searched for two named values it contains: **<default-operation>** and **<test-option>**. Create a regular expression to get the contents of the XML root content in the **<rpc>** tag and then add additional regular expressions to drill down into the content in order to find the value of the **<edit-config>**, **<default-operation>**, and **<test-option>** elements.

```
ns = re.match('{.*}', root.tag).group(0)
editconf = root.find("{}edit-config".format(ns))
defop = editconf.find("{}default-operation".format(ns))
testop = editconf.find("{}test-option".format(ns))
```

e. Add print statements to print the value of the **<default-operation>** and **<test-option>** elements.

```
print("The default-operation contains: {}".format(defop.text))
print("The test-option contains: {}".format(testop.text))
```

Step 2: Run the script.

Save and run the **parsexml.py**. You should get the following output.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsexml.py
The default-operation contains: merge
The test-option contains: set
devasc@labvm:~/labs/devnet-src/parsing$
```

Part 3: Parse JSON in Python

Parsing JavaScript Object Notation (JSON) is a frequent requirement of interacting with REST APIs. The steps are usually as follows:

1) Authenticate using a user/password combination to retrieve a token that will expire after a set amount of time. This token is used for authenticating subsequent requests.
2) Execute a GET request to the REST API, authenticating as required, to retrieve the state of a resource, requesting JSON as the output format.
3) Modify the returned JSON, as needed.
4) Execute a POST (or PUT) to the same REST API (again, authenticating as required) to change the state of the resource, again requesting JSON as the output format and interpreting it as needed to determine whether the operation was successful.

The JSON example to parse is this response from a token request:

```
{

"access_token":"ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItY
TU3",
 "expires_in":1209600,

"refresh_token":"MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1
Njc4",
 "refreshtokenexpires_in":7776000
}
```

In Python scripts, the Python **json** library can be used to parse JSON into Python native data structures, and serialize data structures back out as JSON. The Python **yaml** library can be used to convert the data to YAML.

The following program uses both modules to parse the above JSON data, extract and print data values, and output a YAML version of the file. It uses the **json** library **loads()** method to parse a string into which the file has been read. It then uses normal Python data references to extract values from the resulting Python data structure. Finally, it uses the **yaml** library **dump()** function to serialize the Python data back out as YAML, to the terminal.

Step 1: Build a script to parse the JSON data.

a. Open the **parsejson.py** file found in the **~/labs/devnet-src/parsing** directory.

b. Import the **json** and **yaml** libraries.

```
import json
import yaml
```

c. Use the Python **with** statement to open **myfile.json** and set it to the variable name **json_file**. Then use the **json.load** method to load the JSON file into a string set to the variable name **ourjson**.

Note: There is no need to explicitly close the file as the **with** statement ensures proper opening and closing of the file.

```
with open('myfile.json','r') as json_file:
    ourjson = json.load(json_file)
```

d. Add a print statement for **ourjson** to see that it is now a Python dictionary.

```
print(ourjson)
```

Step 2: Run the script to print the JSON data and then modify it to print data of interest.

a. Save and run your script. You should see the following output.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsejson.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
'expires_in': 1209600, 'refresh_token':
'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4',
'refreshtokenexpires_in': 7776000}
devasc@labvm:~/labs/devnet-src/parsing$
```

b. Add print statements that display the token value and how many seconds until the token expires.

```
print("The access token is: {}".format(ourjson['access_token']))
print("The token expires in {} seconds.".format(ourjson['expires_in']))
```

c. Save and run your script. You should see the following output.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsejson.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
'expires_in': 1209600, 'refresh_token':
'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4',
'refreshtokenexpires_in': 7776000}
1209600
The access token is ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
The token expires in 1209600 seconds
devasc@labvm:~/labs/devnet-src/parsing$
```

Step 3: Output the parsed JSON data in a YAML data format.

a. Add a print statement that will display the three dashes required for a YAML file. The two **\n** will add two lines after the previous output. Then add a statement to print **ourjson** as YAML data by using the **dump()** method of the **yaml** library.

```
print("\n\n---")
print(yaml.dump(ourjson))
```

b. Save and run your script. You should see the following output.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsejson.py
<output from previous steps omitted>
---
access_token: ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
expires_in: 1209600
refresh_token: MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4
refreshtokenexpires_in: 7776000

devasc@labvm:~/labs/devnet-src/parsing$
```

Part 4: Parse YAML in Python

The following program imports the **json** and **yaml** libraries, uses PyYAML to parse a YAML file, extract and print data values, and output a JSON version of the file. It uses the **yaml** library **safe_load()** method to parse the file stream and normal Python data references to extract values from the resulting Python data structure. It then uses the **json** library **dumps()** function to serialize the Python data back out as JSON.

The YAML example to parse is the same YAML file you outputted in Part 3:

```
---
```

```
access_token: ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
expires_in: 1209600
refresh_token: MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4
refreshtokenexpires_in: 7776000
```

Step 1: Build a script to parse the YAML data.

a. Open the **parseyaml.py** file found in the **~/labs/devnet-src/parsing** directory.

b. Import the **json** and **yaml** libraries.

```
import json
import yaml
```

c. Use the Python **with** statement to open **myfile.yaml** and set it to the variable name **yaml_file**. Then use the **yaml.safe_load** method to load the YAML file into a string set to the variable name **ouryaml**.

```
with open('myfile.yaml','r') as yaml_file:
    ouryaml = yaml.safe_load(yaml_file)
```

d. Add a print statement for **ouryaml** to see that it is now a Python dictionary.

```
print(ouryaml)
```

Step 2: Run the script to print the YAML data and then modify it to print data of interest.

a. Save and run your script. You should see the following output.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
'expires_in': 1209600, 'refresh_token':
'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4',
'refreshtokenexpires_in': 7776000}
devasc@labvm:~/labs/devnet-src/parsing$
```

b. Add print statements that display the token value and how many seconds until the token expires.

```
print("The access token is {}".format(ouryaml['access_token']))
print("The token expires in {} seconds.".format(ouryaml['expires_in']))
```

c. Save and run your script. You should see the following output.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
'expires_in': 1209600, 'refresh_token':
'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4',
'refreshtokenexpires_in': 7776000}
The access token is ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
The token expires in 1209600 seconds.
devasc@labvm:~/labs/devnet-src/parsing$
```

Step 3: Output the parsed YAML data in a JSON data format.

a. Add a print statement to add two blank lines after the previous output. Then add a statement to print **ouryaml** as JSON data by using the **dumps()** method of the **json** library. Add the indent parameter to prettify the JSON data.

```
print("\n\n")
print(json.dumps(ouryaml, indent=4))
```

b. Save and run your script. You should see the following output. Notice that the output looks just like the **myfile.json**.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
<output from previous steps omitted>
{
  "access_token": "ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3",
  "expires_in": 1209600,
  "refresh_token": "MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4",
  "refreshtokenexpires_in": 7776000
}
devasc@labvm:~/labs/devnet-src/parsing$
```