

UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE CIENCIAS



ÁREA DE CIENCIA DE LA COMPUTACIÓN

3° LABORATORIO - CC312

2 PARTE

- **TÍTULO:** ANALIZAR DIFERENTES TIPOS DE DATOS CON PYTHON
- **ALUMNO:**
JHONATAN POMA MARTINEZ 20182729F
- **PROFESORES:** YURI JAVIER.,CCOICCA PACASI

2023

OBJETIVOS:

Parte 1: Iniciar la máquina virtual (Virtual Machine) de DEVASC.

Parte 2: Analizar XML en Python.

Parte 3: Analizar JSON en Python.

Parte 4: Analizar YAML en Python.

INTRODUCCIÓN:

Examinar significa analizar un mensaje, dividiéndolo en sus partes componentes y comprender el propósito de cada parte en contexto. Cuando los mensajes se transmiten entre equipos, viajan como una secuencia de caracteres. Esos caracteres son efectivamente una string (cadena). Ese mensaje debe ser analizado en una estructura de datos semánticamente equivalente que contenga datos de tipos reconocidos (por ejemplo, integers, floats, strings, and booleans) antes de que los datos puedan ser interpretados y actuados sobre ellos.

En este laboratorio, usará Python para analizar cada formato de datos a su vez: XML, JSON y YAML.

PARTE 1: Iniciar la máquina virtual (Virtual Machine) de DEVASC.

PARTE 2: Analizar XML en Python

XML: significa lenguaje de marcado extensible (eXtensible Markup Language). Fue diseñado para almacenar y transportar datos. Fue diseñado para ser legible tanto por humanos como por máquinas. Por eso, los objetivos de diseño de XML enfatizan la simplicidad, la generalidad y la facilidad de uso en Internet.

XML tree and elements (árbol XML y elementos):

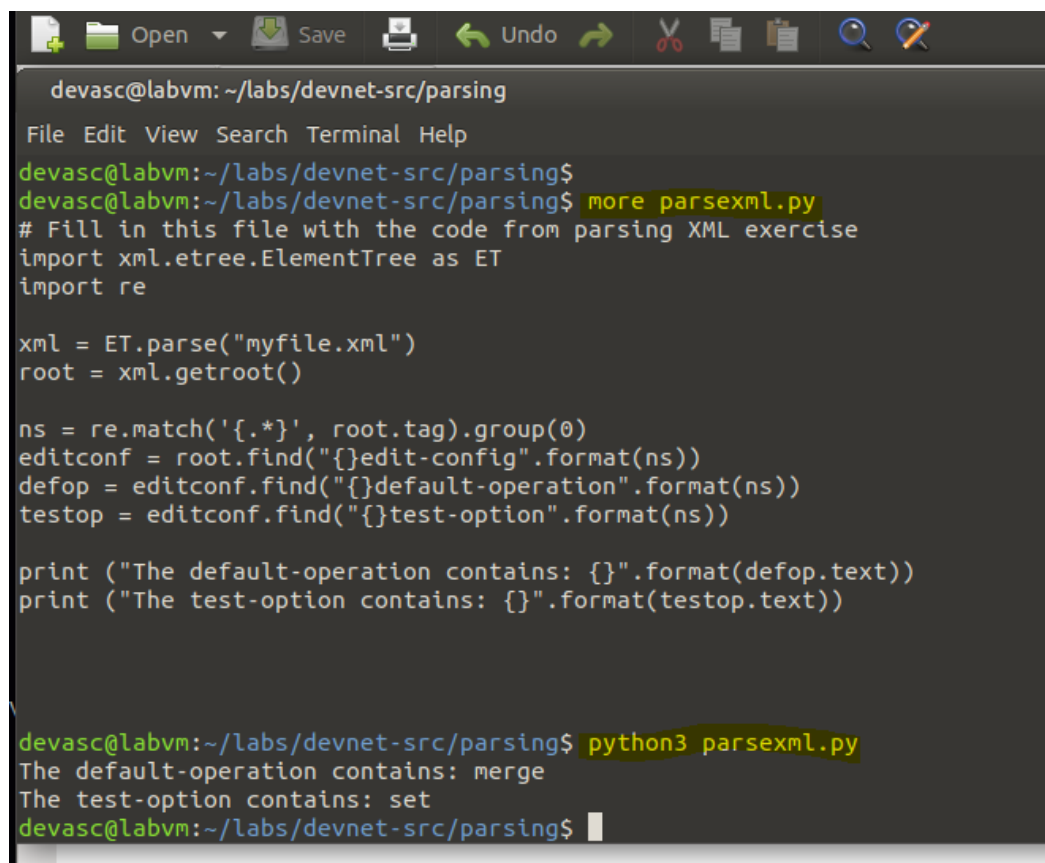
XML es un formato de datos inherentemente jerárquico y la forma más natural de representarlo es con un árbol. “**ET**” tiene dos clases para este propósito: **ElementTree** presenta todo el documento XML como un árbol y **Element** representa un solo nodo en este árbol. Las interacciones con todo el documento (leer y escribir en/desde archivos) generalmente se realizan en el nivel **ElementTree**. Las interacciones con un solo elemento XML y sus subelementos se realizan en el nivel **Element**.

Mostramos el archivo **myfile.xml**



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rpc message-id="1"
3   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
4   <edit-config>
5     <target>
6       <candidate/>
7     </target>
8     <default-operation>merge</default-operation>
9     <test-option>set</test-option>
10    <config>
11      <int8.1
12        xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
13        nc:operation="create"
14        xmlns="http://netconfcentral.org/ns/test">9</int8.1>
15      </config>
16    </edit-config>
17  </rpc>
```

Creamos un script y lo ejecutamos para Analizar los datos XML.



```
devasc@labvm: ~/labs/devnet-src/parsing
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/parsing$ more parsexml.py
# Fill in this file with the code from parsing XML exercise
import xml.etree.ElementTree as ET
import re

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

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))

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

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$
```

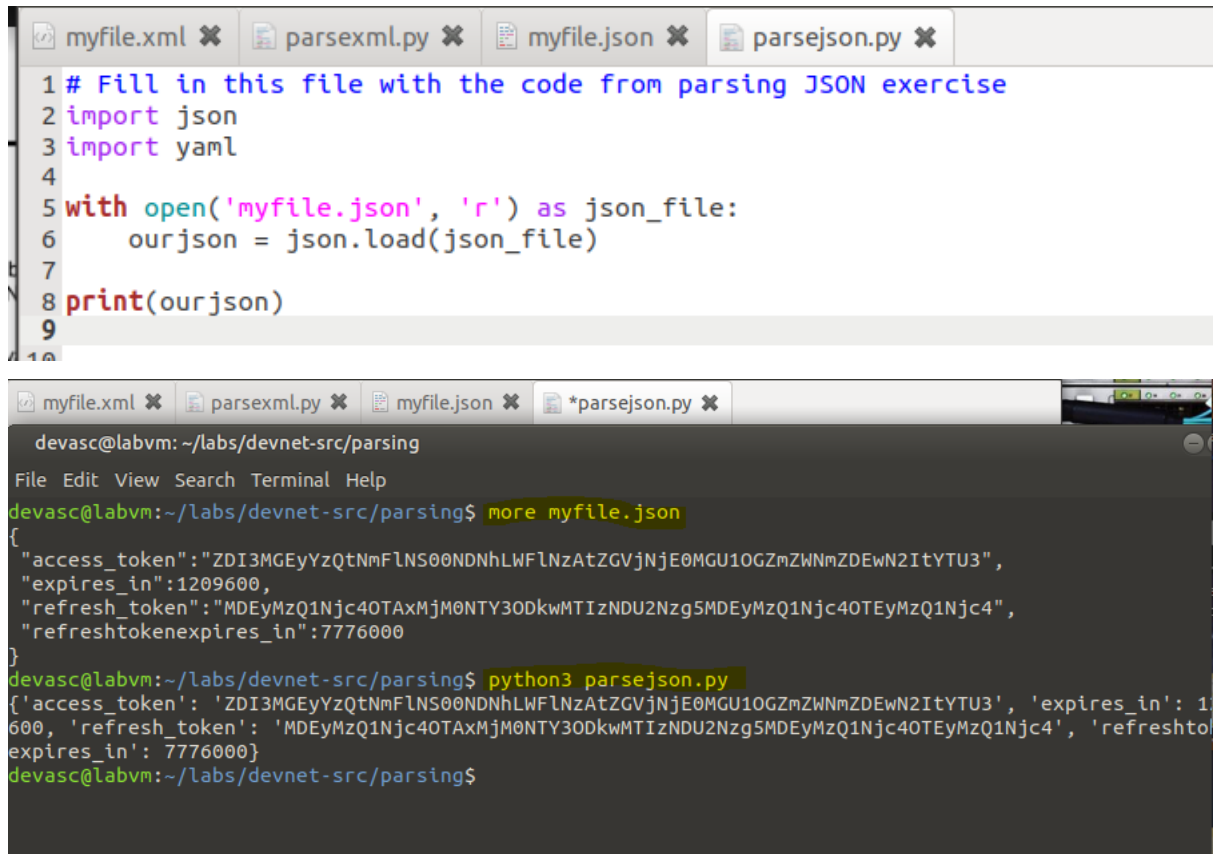
ElementTree se utilizará para realizar el análisis.

Parte 3: Analizar JSON en Python.

JSON es una sintaxis para almacenar e intercambiar datos.

JSON es texto, escrito con notación de objetos JavaScript.

Crearemos un script para analizar los datos JSON.



The screenshot shows a code editor with four tabs: myfile.xml, parsexml.py, myfile.json, and parsejson.py. The parsejson.py file contains the following Python code:

```
1 # Fill in this file with the code from parsing JSON exercise
2 import json
3 import yaml
4
5 with open('myfile.json', 'r') as json_file:
6     ourjson = json.load(json_file)
7
8 print(ourjson)
9
```

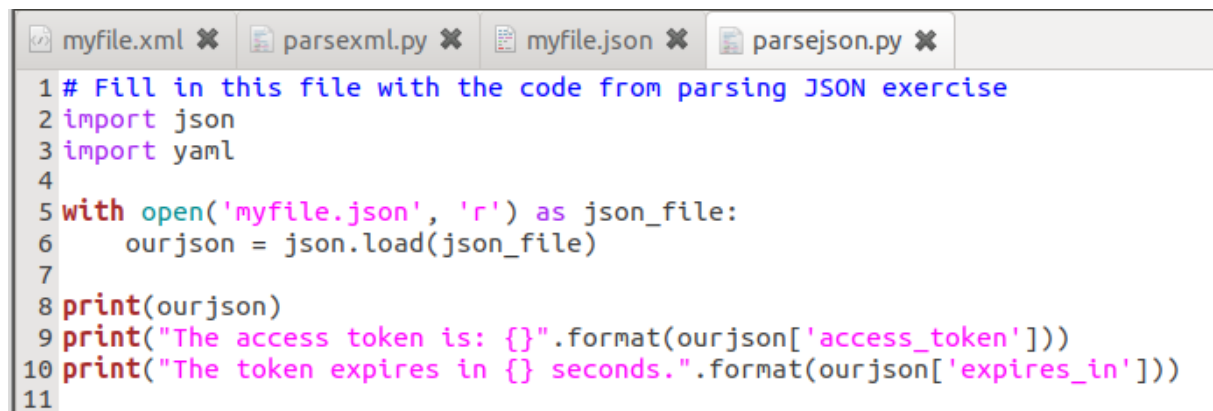
Below the code editor is a terminal window showing the execution of the script. The terminal prompt is devasc@labvm: ~/labs/devnet-src/parsing. The user runs the command `more myfile.json`, which displays the contents of the JSON file:

```
{
  "access_token": "ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3",
  "expires_in": 1209600,
  "refresh_token": "MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4",
  "refresh_tokenexpires_in": 7776000
}
```

The user then runs the command `python3 parsejson.py`, which outputs the parsed JSON data as a Python dictionary:

```
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3', 'expires_in': 1209600, 'refresh_token': 'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4', 'refresh_tokenexpires_in': 7776000}
```

Agregaremos dos instrucciones en nuestro script para que muestre el acceso del token y el tiempo antes que caduque el token.



The screenshot shows the same code editor with the parsejson.py file updated to include two additional print statements. The code is as follows:

```
1 # Fill in this file with the code from parsing JSON exercise
2 import json
3 import yaml
4
5 with open('myfile.json', 'r') as json_file:
6     ourjson = json.load(json_file)
7
8 print(ourjson)
9 print("The access token is: {}".format(ourjson['access_token']))
10 print("The token expires in {} seconds.".format(ourjson['expires_in']))
11
```

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

mandaremos por impresión los datos en un formato YAML.

```
myfile.xml x parsexml.py x myfile.json x parsejson.py x
1 # Fill in this file with the code from parsing JSON exercise
2 import json
3 import yaml
4
5 with open('myfile.json', 'r') as json_file:
6     ourjson = json.load(json_file)
7
8 print(ourjson)
9 print("The access token is: {}".format(ourjson['access_token']))
10 print("The token expires in {} seconds.".format(ourjson['expires_in']))
11
12 print("\n\n---")
13 print(yaml.dump(ourjson))
```

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

---
access_token: ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
expires_in: 1209600
refresh_token: MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4
refresh_tokenexpires_in: 7776000
devasc@labvm:~/labs/devnet-src/parsing$
```

Parte 4: Analizar YAML en Python.

YAML, un acrónimo recursivo de "YAML Ain't Markup Language", es un lenguaje de serialización de datos legible por humanos. A menudo se usa para archivos de configuración, pero también para el intercambio de datos. El analizador YAML de Python más utilizado es PyYAML, una biblioteca que permite cargar, analizar y escribir YAML.

El siguiente programa importa las librerías `json` y `yaml`, utiliza `PyYaml` para analizar un archivo YAML, extraer e imprimir valores de datos y generar una versión JSON del archivo.

Utilizar el método de librería **`yamlsafe_load()`** para analizar la secuencia de archivos y las referencias normales de datos de Python para extraer valores de la estructura de datos de Python resultante.

```
devasc@labvm:~/labs/devnet-src/parsing$ more myfile.yaml
---
access_token: ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
expires_in: 1209600
refresh_token: MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4
refreshtokenexpires_in: 7776000
devasc@labvm:~/labs/devnet-src/parsing$
```

Creamos un script para analizar los datos de YAML

```
myfile.xml x parsexml.py x myfile.json x parsejson.py x parseyaml.py x
1 # Fill in this file with the code from parsing YAML exercise
2 import json
3 import yaml
4
5 with open('myfile.yaml','r') as yaml_file:
6     ouryaml = yaml.safe_load(yaml_file)
7
8 print(ouryaml)
9
```

Ejecutamos el script.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
 'expires_in': 1209600, 'refresh_token': 'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4', 'refreshtokenexpires_in': 7776000}
devasc@labvm:~/labs/devnet-src/parsing$
```

```
myfile.xml x parsexml.py x myfile.json x parsejson.py x parseyaml.py x
1 # Fill in this file with the code from parsing YAML exercise
2 import json
3 import yaml
4
5 with open('myfile.yaml', 'r') as yaml_file:
6     ouryaml = yaml.safe_load(yaml_file)
7
8 print(ouryaml)
9
10 print("The access token is {}".format(ouryaml['access_token']))
11 print("The token expires in {} seconds.".format(ouryaml['expires_in']))
```

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

Mostramos los datos YAML analizados en un formato de datos JSON

```
myfile.xml x parsexml.py x myfile.json x parsejson.py x parseyaml.py x
1 # Fill in this file with the code from parsing YAML exercise
2 import json
3 import yaml
4
5 with open('myfile.yaml', 'r') as yaml_file:
6     ouryaml = yaml.safe_load(yaml_file)
7
8 print(ouryaml)
9
10 print("The access token is {}".format(ouryaml['access_token']))
11 print("The token expires in {} seconds.".format(ouryaml['expires_in']))
12
13 print("\n\n--")
14 print(json.dumps(ouryaml, indent=4))
15
```

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

--
{
  "access_token": "ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3",
  "expires_in": 1209600,
  "refresh_token": "MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4",
  "refresh_tokenexpires_in": 7776000
}
devasc@labvm:~/labs/devnet-src/parsing$
```


References

Expresiones regulares COMOS (HOWTO) — documentación de Python - 3.11.3. (n.d.). Python Docs. Retrieved April 16, 2023, from <https://docs.python.org/es/3/howto/regex.html>

Rahman, S. (2022, April 6). *JSON vs YAML vs XML — when to use what?* Medium. Retrieved April 16, 2023, from <https://medium.com/black-book-for-data/json-vs-yaml-vs-xml-when-to-use-what-1994d4448335>

xml.etree.ElementTree — The ElementTree XML API — Python 3.11.3 documentation. (n.d.). Python Docs. Retrieved April 16, 2023, from <https://docs.python.org/3/library/xml.etree.elementtree.html>

YAML vs. JSON: What is the difference? (2023, March 20). Imaginary Cloud. Retrieved April 16, 2023, from <https://www.imaginarycloud.com/blog/yaml-vs-json-what-is-the-difference/>