04_introduccion_a_yaml

May 16, 2023

1 Introducción a *YAML*.

YAML es el acrónimo de "*Yet Another Markup Language*". Es un lenguaje derivado de *JSON* que es utilizado para la serialización de datos, como lenguaje declarativ e incluso es posible desarrolla lenguajes de dominio específico (*DSL*) . Los archivos de *YAML* tiene la extensión `.yml` o `.yaml`.

YAML, al igual que *JSON* es utilizado para la serialización de estructuras de datos. Los documentos utilizando el formato de *YAML* tiene la ventaja de que que pueden ser procesados eficientemente por sistemas de cómputo y al mismo tiempo son muy legibles para los seres humanos.

Algunas herramientas que consumen documentos en formato *YAML* son entre otros:

- Los archivos *compose* de *Docker Compose*.
- Los *Playbooks* de *Ansible*.
- Las definiciones de *Kubernetes*.
- Los *workflows* de *GitHub Actions*.

Para mayor referencia es posible consultar la siguiente liga:

(https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html).

2 Tipo de datos.

Todos los valores definidos en un documento en formato *YAML* son considerados como cadenas de caracteres. El sistema que consuma los archivos *YAML* es el encargado de inferir el tipo de dato del que se trata.

2.1 Sintaxis de *YAML*.

El componente básico de una estructura de datos en *YAML* son:

- Mapas.
- Listas.

A diferencia de *JSON* no se uitilizan las llaves `{ }` para delimitar un bloque de datos, sino que de forma similar a *Python*, se utiliza la indentación.

NOTA: *YAML* no acepta el uso de tabuladores para la indentación. Sólo se aceptan espacios.

2.2 Mapas.

Los mapas son pares `<clave>: <valor>`, donde el valor puede ser: * Una cadena de caracteres. * Un mapa. * Una lista.

Ejemplos:

El siguiente texto contiene a los siguientes pares:

- La clave correspondiente a `nombre` y el valor correspondiente a `Juan`.
- La clave correspondiente a `ruta` y el valor correspondiente a `"/valr/lib/docker"`.

```
nombre: Juan
ruta: "/var/lib/docker"
```

2.2.1 Mapas anidados.

Los mapas pueden contener a su vez otras estrcuturas de mapas. Para indicar que un mapa pertenece a otro se utiliza la indentación.

```
<clave 1> :
    <clave 11>: <valor 11>
    <clave 12>: <valor 12>
    ...
    ...
    <clave 1n>: <valor 1n>
```

Ejemplo:

El siguiente texto define al mapa con clave `persona`, el cual contiene a los mapas con claves:

- `id`.
- `nombre`.
- `primer_apellido`.
- `segundo_apellido`.
- `carrera`.

```
persona:
    id: 1001
    nombre: Juan
    primer_apellido: Pérez
    segundo_apellido: de la Rosa
    carrera: Derecho
```

2.3 Listas.

las listas se definen mediante columnas en las que cada elemento de dicha lista es precedido por un guión medio `-`.

```
- <elemento 1>
- <elemento 2>
...
...
```

- <elemento n>

Estos elementos pueden ser cadenas de caracteres, mapas u otras listas.

Ejemplo:

El siguiente texto define una estructura que incluye mapas y listas anidadas dentro de una lista.

```
cursos:
 - py101: AC
 - py111: AC
 - py121: NA
 - py131:
  programado:
   clave_evento: py131-2007
   fecha_inicio: 15/07/2020
   requisitos:
    - py101
    - py111
 - py201: NA
```

2.4 Serialización con *YAML* con *Python*.

De forma similar a *JSON*, *YAML* permite serializar estructuras de datos complejas.

Para ilustrar la serialización, se utilizará la biblioteca `pyyaml`.

[ ]:
```
pip install pyyaml
```

[ ]:
```
import yaml
```

- La siguiente celda creará un objeto de tipo `dict` de *Python* llamado `listado`, el cual contiene una estructura de datos compleja.

[ ]:
```
listado = {'personas': [
    {"id": 1001,
    'nombre': 'Juan',
    'primer_apellido': 'Pérez',
    'segundo_apellido': 'de la Rosa',
    'carrera': 'Derecho',
    'cursos': {
        'py101': 'AC',
        'py111': 'AC',
        'py121': 'NA',
        'py131': {
            'PN': {
                'clave_evento': 'py131-2007',
                'fecha_inicio': '15/07/2020',
                'requisitos': [
                    'py101',
                    'py111']}},
```

```
        'py201': 'NA'}},
    {"id": 1002,
     'nombre': 'María',
     'primer_apellido': 'Mendoza',
     'segundo_apellido': '',
     'carrera': 'Derecho',
     'cursos': {
         'py101': 'AC',
         'py111': 'AC',
         'py121': 'AC'}}]}
```

- Los objetos `dict` de *Python* permiten realizar indexación por claves, por lo que la siguiente celda traerá el objeto con índice `personas` del objeto `listado`. En este caso, es un objeto de tipo `list`.

[ ]:
```
listado['personas']
```

- La siguiente celda traerá el primer elemento del objeto `listado['personas']`.

[ ]:
```
listado['personas'][0]
```

- La función `yaml.dump()` permite serializar un objeto de *Python* en formato *YAML*. El resultado es uan representación en binario del objeto.

[ ]:
```
yaml.dump(listado, encoding="utf-8")
```

- La siguient celda le asignará el nombre `estructura` al objeto `bytes` que contiene la serialización del objeto `listado` en formato *YAML*.

[ ]:
```
estructura = yaml.dump(listado, encoding="utf-8")
```

- La función `yaml.dump()` también permite escribir en archivos el contenido serializado.

[ ]:
```
with open("archivo.yaml", "w") as f:
    yaml.dump(listado, f)
```

- El siguiente es el contenido del archivo `archivo.yaml`.

[ ]:
```
!cat archivo.yaml
```

- La función `yaml.load()` permite construir un objeto nuevo de *Python* a partir de una cadena de caracteres en formato *YAML*.

[ ]:
```
otro_listado = yaml.load(estructura, yaml.Loader)
```

[ ]:
```
otro_listado
```

[ ]:
```
listado == otro_listado
```

[ ]:
```
listado is otro_listado
```