

## XML

XML significa Extensible Markup Language y está diseñado para transmitir y almacenar datos.

es un conjunto de reglas para definir la semántica de etiquetas, estas etiquetas documentarán divididos en muchas partes y estas partes podrán ser identificadas, también es un lenguaje de meta marcado que define la sintaxis del lenguaje utilizado para definir otra, semántica, lenguaje de marcado con estructura de dominio específico.

## DOM (Document Object Model)

El Modelo de Objetos del Documento, o DOM por sus siglas en inglés, es un lenguaje API del Consorcio World Wide Web (W3C) para acceder y modificar documentos XML. Una implementación del DOM presenta los documentos XML como un árbol, o permite al código cliente construir dichas estructuras desde cero para luego darles acceso a la estructura a través de un conjunto de objetos que implementaron interfaces conocidas.

El DOM es extremadamente útil para aplicaciones de acceso directo. SAX sólo te permite la vista de una parte del documento a la vez. Si estás mirando un elemento SAX, no tienes acceso a otro. Si estás viendo un nodo de texto, no tienes acceso al elemento contenedor. Cuando desarrollas una aplicación SAX, necesitas registrar la posición de tu programa en el documento en algún lado de tu código. SAX no lo hace por ti. Además, desafortunadamente no podrás mirar hacia adelante (look ahead) en el documento XML.

Algunas aplicaciones son imposibles en un modelo orientado a eventos sin acceso a un árbol. Por supuesto que puedes construir algún tipo de árbol por tu cuenta en eventos SAX, pero el DOM te evita escribir ese código. El DOM es una representación de árbol estándar para datos XML.

El Modelo de Objetos del Documento es definido por el W3C en fases, o niveles en su terminología. El mapeado de Python de la API está basado en la recomendación del DOM nivel 2.

Las aplicaciones DOM típicamente empiezan al diseccionar (parse) el XML en un DOM. Cómo esto funciona no está incluido en el DOM nivel 1, y el nivel 2 provee mejoras limitadas. Existe una clase objeto llamada DOMImplementation que da acceso a métodos de creación de Document, pero de ninguna forma da acceso a los constructores (builders) de reader/parser/Document de una forma independiente a la implementación. No hay una forma clara para acceder a estos métodos sin un objeto Document existente. En Python, cada implementación del DOM proporcionará una función `getDOMImplementation()`. El DOM de nivel 3 añade una especificación para Cargar(Load)/Guardar(Store), que define una interfaz al lector (reader), pero no está disponible aún en la librería estándar de Python.

El módulo `xml.dom` contiene las siguientes funciones:

`xml.dom.registerDOMImplementation(name, factory)`

Registra la función `factory` con el nombre `name`. La función fábrica (`factory`) debe retornar un objeto que implemente la interfaz `DOMImplementation`. La función fábrica puede retornar el mismo objeto cada vez que se llame, o uno nuevo por cada llamada, según sea apropiado para la implementación específica (e.g. si la implementación soporta algunas personalizaciones).

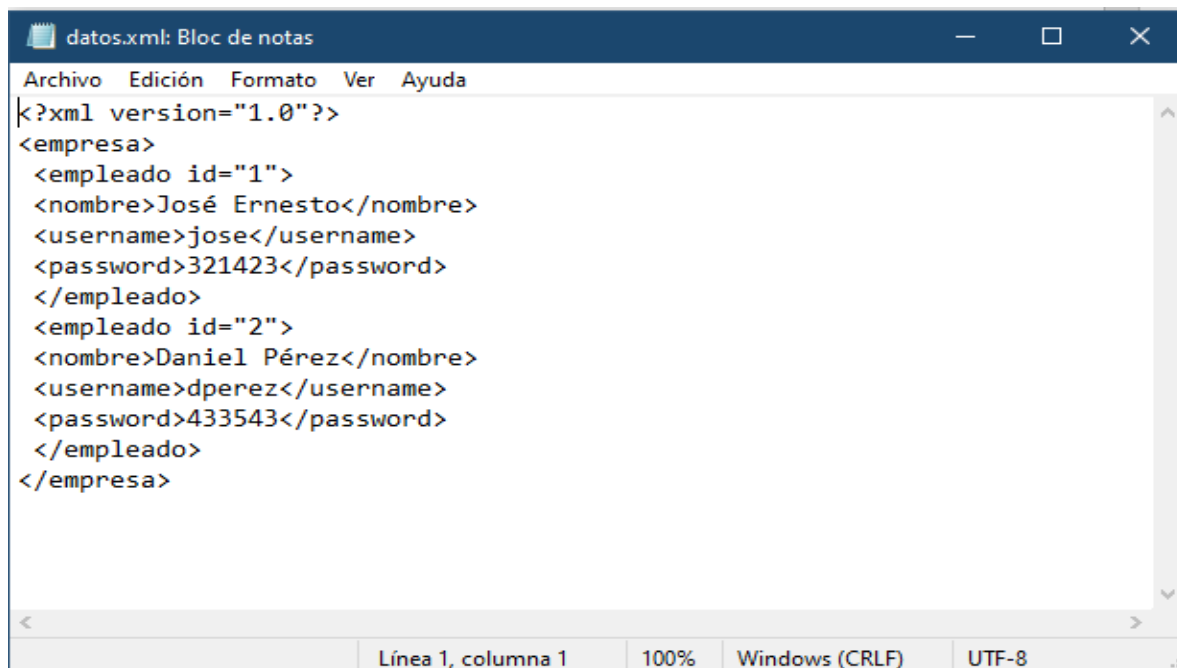
`xml.dom.getDOMImplementation(name=None, features=())`

Retorna una implementación del DOM apropiada. El `name` es o bien conocido, el nombre del módulo de una implementación DOM, o `None`. Si no es `None` importa el módulo correspondiente y retorna un objeto `DomImplementation` si la importación tiene éxito. Si no se le pasa un nombre, y el entorno de variable `PYTHON_DOM` ha sido puesto, dicha variable es usada para encontrar la información de la implementación.

Si no se le pasa un nombre, examina las implementaciones disponibles para encontrar uno con el conjunto de características requeridas. Si no se encuentra ninguna implementación, levanta una excepción `ImportError`. La lista de características debe ser una secuencia de pares (`feature,version`) que son pasados al método `hasFeature()` en objetos disponibles de `DOMImplementation`.

Podemos encontrar diferentes funciones y objetos para el módulo `xml.dom` así que a continuación se muestra un ejemplo del uso de `xml.dom`:

Primero se crea o se obtiene un archivo `.xml` y en este caso se tiene archivo con la siguiente información:



```
datos.xml: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
<?xml version="1.0"?>
<empresa>
  <empleado id="1">
    <nombre>José Ernesto</nombre>
    <username>jose</username>
    <password>321423</password>
  </empleado>
  <empleado id="2">
    <nombre>Daniel Pérez</nombre>
    <username>dperez</username>
    <password>433543</password>
  </empleado>
</empresa>
```

Línea 1, columna 1    100%    Windows (CRLF)    UTF-8

Ya en PyCharm realizamos el siguiente código de Python:

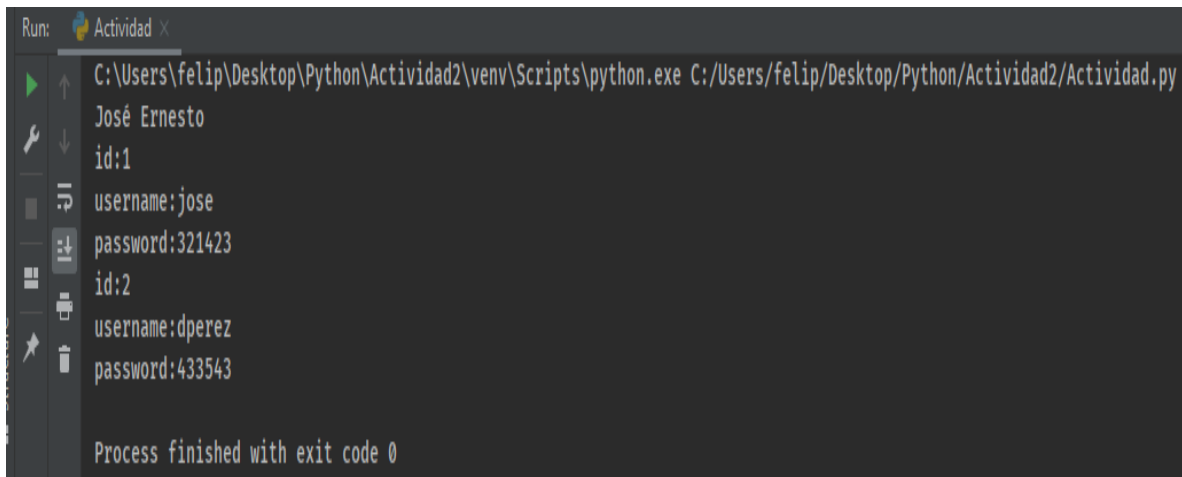
```
1  from xml.dom import minidom
2
3  doc = minidom.parse("C:/Users/felip/Desktop/datos.xml")
4  nombre = doc.getElementsByTagName("nombre")[0]
5  print(nombre.firstChild.data)
6  empleados = doc.getElementsByTagName("empleado")
7  for empleado in empleados:
8      sid = empleado.getAttribute("id")
9      username = empleado.getElementsByTagName("username")[0]
10     password = empleado.getElementsByTagName("password")[0]
11     print("id:%s " % sid)
12     print("username:%s" % username.firstChild.data)
13     print("password:%s" % password.firstChild.data)
```

La función `parse()` puede tomar un nombre de archivo o un objeto de archivo previamente abierto.

`minidom.parse(filename_or_file, parser=None, bufsize=None)`

Retorna un Document a partir de la entrada dada. `filename_or_file` puede ser un nombre de archivo o un objeto similar a un archivo. `parser`, si se proporciona, debe ser un objeto de un analizador sintáctico SAX2. Esta función intercambiará el controlador de documentos del analizador sintáctico y activará el soporte con el espacio de nombres. Otras configuraciones del analizador sintáctico (como configurar un solucionador de entidades) deben haberse realizado de antemano.

Y el código nos imprime lo siguiente en consola:



```
Run: Actividad x
C:\Users\felip\Desktop\Python\Actividad2\venv\Scripts\python.exe C:/Users/felip/Desktop/Python/Actividad2/Actividad.py
José Ernesto
id:1
username:jose
password:321423
id:2
username:dperez
password:433543

Process finished with exit code 0
```

Otro ejemplo sería el siguiente:

```
1  import xml.dom.minidom
2
3  document = """\
4  <slideshow>
5  <title>Demo slideshow</title>
6  <slide><title>Slide title</title>
7  <point>This is a demo</point>
8  <point>Of a program for processing slides</point>
9  </slide>
10
11  <slide><title>Another demo slide</title>
12  <point>It is important</point>
13  <point>To have more than</point>
14  <point>one slide</point>
15  </slide>
16  </slideshow>
17  """
18
19  dom = xml.dom.minidom.parseString(document)
20
21  def getText(nodelist):
22      rc = []
23      for node in nodelist:
24          if node.nodeType == node.TEXT_NODE:
25              rc.append(node.data)
26      return ''.join(rc)
27
28  def handleSlideshow(slideshow):
29      print("<html>")
30      handleSlideshowTitle(slideshow.getElementsByTagName("title")[0])
31      slides = slideshow.getElementsByTagName("slide")
32      handleToc(slides)
33      handleSlides(slides)
34      print("</html>")
35
36  def handleSlides(slides):
37      for slide in slides:
38          handleSlide(slide)
39
40  def handleSlide(slide):
41      handleSlideTitle(slide.getElementsByTagName("title")[0])
42      handlePoints(slide.getElementsByTagName("point"))
43
```

```

44 def handleSlideshowTitle(title):|
45     print("<title>%s</title>" % getText(title.childNodes))
46
47 def handleSlideTitle(title):
48     print("<h2>%s</h2>" % getText(title.childNodes))
49
50 def handlePoints(points):
51     print("<ul>")
52     for point in points:
53         handlePoint(point)
54     print("</ul>")
55
56 def handlePoint(point):
57     print("<li>%s</li>" % getText(point.childNodes))
58
59 def handleToc(slides):
60     for slide in slides:
61         title = slide.getElementsByTagName("title")[0]
62         print("<p>%s</p>" % getText(title.childNodes))
63
64     handleSlideshow(dom)

```

Lo que en consola imprime lo siguiente:

```

Run: Ejemplo2 x
C:\Users\felip\Desktop\Python\Actividad2\venv\Scripts\python.exe C:/Users/felip/Desktop/Python/Actividad2/Ejemplo2.py
<html>
<title>Demo slideshow</title>
<p>Slide title</p>
<p>Another demo slide</p>
<h2>Slide title</h2>
<ul>
<li>This is a demo</li>
<li>Of a program for processing slides</li>
</ul>
<h2>Another demo slide</h2>
<ul>
<li>It is important</li>
<li>To have more than</li>
<li>one slide</li>
</ul>
</html>

Process finished with exit code 0

```

## **Xpath**

El lenguaje de ruta XML (XPath) es una herramienta enormemente subestimada en el mundo del web scraping y la automatización. Provee una serie de expresiones para localizar elementos en un árbol, su finalidad es proporcionar un conjunto de sintaxis, por lo que debido a su limitado alcance no se considera un motor en si mismo.

Cada elemento de una página web está organizado por el Modelo de objetos de documento (DOM). El DOM es una estructura en forma de árbol, donde cada elemento representa un nodo, con rutas a los nodos padre e hijo.

XPath nos ofrece un lenguaje para atravesar rápidamente este árbol. Y, podemos agregar lógica a nuestra selección de nodos para hacer nuestras consultas más poderosas.