UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

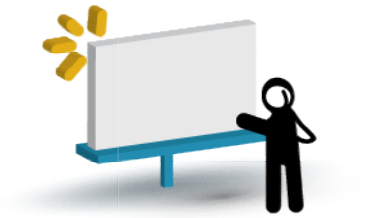
UNIDAD DIDÁCTICA VII

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo VI – Nivel Avanzado

Unidad VII – XML y JSON.



Presentación:

XML (Extensible Markup Language) es un lenguaje de etiquetas extensible, que fue diseñado principalmente para formatear y transferir datos entre distintas plataformas y lenguajes. Es el lenguaje del cual deriva html, de hecho html es un caso particular de xml en donde las etiquetas que se utilizan son fijas. XML es el lenguaje actualmente utilizado por plataformas como android para crear las vistas de las aplicaciones. En esta unidad veremos cómo integrar xml a python de forma de leer, escribir y modificar datos de archivos xml desde python.

También trabajaremos con el formato JSON, el cual en aplicaciones móviles, videojuegos, animaciones y texturas es muy utilizado actualmente.

Objetivos:

Que los participantes:.

Aprendan a trabajar con los lenguajes de transferencia y formateo de datos XML y JSON.

Bloques temáticos:

1.- XML - eXtensible Markup Language.

2.- ElementTree.

3.- minidom.

4.- JSON.

Consignas para el aprendizaje colaborativo

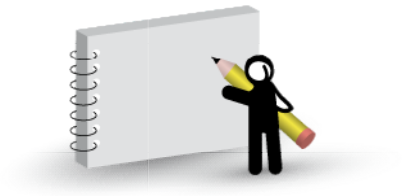
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1.- XML - eXtensible Markup Language

Este lenguaje como su nombre lo indica, es un lenguaje de etiquetas extensibles, es decir que pueden ser creadas etiquetas nuevas según las necesidades de cada desarrollo. Una etiqueta es un nombre entre símbolos de menor y mayor y podemos tener etiquetas que requieren un cierre y otras que no:

- Cómo ejemplo de una etiqueta con apertura y cierre podríamos considerar una etiqueta destinada a almacenar nombres de personas llamada <nombre>, la etiqueta de cierre debe anteponer al nombre un slash o barra inclinada </nombre>. El contenido de la etiqueta no es analizado sintácticamente y son considerados solo datos de carácter, mientras que la etiqueta en sí es analizada sintacticamente.

```
<nombre>Maximiliano</nombre>
```

- Cómo ejemplo de una etiqueta que no requiere cierre podemos considerar la etiqueta utilizada para representar un salto de línea
, en este caso la barra inclinada se escribe antes del signo de mayor, aún cuando podría ser omitida. Las etiquetas que no requieren cierre se suelen llamar vacías.

Encabezada

El documento xml comienza con una declaración de tipo de documento, en donde se especifica la versión (indicada en rojo), el tipo de codificación (indicado en azul) y la declaración standalone (en verde). Los dos valores posibles de standalone son yes y no. Un valor yes indica que el documento es autónomo, por lo que no se apoya en fuentes de información externas, incluyendo declaraciones de marcado. Un valor de no indica que el documento no es autónomo, por lo que se puede apoyar en fuentes de información externas. :

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
```

Etiqueta raíz.

Todo documento xml debe tener una etiqueta principal llamada raíz dentro de la cual se escriben el resto de las etiquetas:

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <clase>
3
4    </clase>
```

Atributos

Las etiquetas pueden poseer atributos de forma de describir acciones específicas, estos se ponen dentro de la etiqueta de apertura y su valor se adiciona entre comillas simples o dobles.

Un ejemplo concreto podría ser el siguiente, en donde la etiqueta <clase> es considerada la etiqueta raíz y cada etiqueta posee el atributo “edad” destinado a especificar la edad de la persona que puede ser de sexo masculino o femenino.

xml1.xml

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <clase>
3            <varon edad='5'>
4               Juan
5            </varon>
6            <mujer edad='4'>
7               Ana
8            </mujer>
9    </clase>
```

La utilización de etiquetas no es recomendable cuando:

- Los valores de los atributos contienen cadenas de texto muy largas.
- Se requiere agregar información anidada.

En estos casos es recomendable realizar la arquitectura en base a etiquetas.

Al abrir este documento con un explorador podemos ver que el contenido es interpretado como un diagrama de árbol, en donde las etiquetas se pueden colapsar utilizando el signo menos (-) que aparece a la izquierda de la etiqueta:

Este archivo XML parece no tener información de estilo asociada. El árbol del documento se muestra debajo.

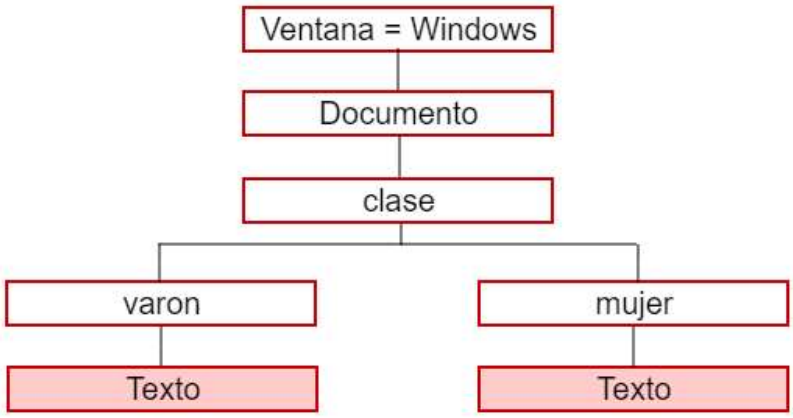
```
−<clase>
    <varon edad="5"> Juan </varon>
    <mujer edad="4"> Ana </mujer>
 </clase>
```

Este archivo XML parece no tener información de estilo asociada. El árbol del documento se muestra debajo.

```
+<clase></clase>
```

Esta forma de entender un documento xml por el explorador se denomina DOM (Document Objet Model) en donde el documento es entendido como una estructura de objetos (nodos) en el momento de cargarlo.

En forma esquemática el explorador entiende al archivo anterior como si fueran las raíces de un árbol, de la siguiente forma:



La distribución de python viene con el módulo ElementTree, el cual nos permite trabajar con código xml.

2.- ElementTree

Utilizar ElementTree es muy similar a minidom, sin embargo el objeto attrib retorna un objeto de diccionario el cual es un poco más simple de utilizar. Al acceder a objetos y atributos utilizamos listas y diccionarios en lugar de nodos del DOM.

Leer archivo

xml_elementtree1.py

```
1   import os
2   import xml.etree.ElementTree as ET
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml1.xml"
4
5   diagrama = ET.parse(archivo)
6   elementoRaiz = diagrama.getroot()
7   print(elementoRaiz)
8   print(elementoRaiz[0].text)
9   print(elementoRaiz[0].attrib)
10  print(elementoRaiz[0].attrib['edad'])
11  print(elementoRaiz[1].text)
12  print(elementoRaiz[1].attrib)
13  print(elementoRaiz[1].attrib['edad'])
```

Nos retorna:

```
<Element 'clase' at 0x036DD990>
        Juan
{'edad': '5'}
5
      Ana
{'edad': '4'}
4
```

Escribir un XML

xml_elementtree2.py

```
1   import os
2   import xml.etree.ElementTree as ET
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml2.xml"
4
5   # Creamos cabecera
6   cabecera = '<?xml version="1.0" encoding="utf-8"?>'
7   # Creamos elementos raíz
8   raiz = ET.Element('mediolocomocion')
9   vehiculo = ET.SubElement(raiz, 'vehiculo')
10  moto = ET.SubElement(vehiculo, 'moto')
11  auto = ET.SubElement(vehiculo, 'auto')
12  # Creamos atributo
13  moto.set('marca','bmw')
14  auto.set('marca','audi')
15  moto.text = 'Moto número 1'
16  auto.text = 'Auto número 1'
17
18  # creamos XML
19  datos = ET.tostring(raiz)
20  datosstr = datos.decode("utf-8")
21  archivo = open(archivo, 'w')
22  archivo.write(cabecera)
23  archivo.write(datosstr)
24  archivo.close()
```

Nos retorna:

```
−<mediolocomocion>
  −<vehiculo>
      <moto marca="bmw">Moto número 1</moto>
      <auto marca="audi">Auto número 1</auto>
    </vehiculo>
</mediolocomocion>
```

Encontrar elementos

El módulo ElementTree nos permite además utilizar algunas funciones a utilizar en la búsqueda de elementos del documento xml:

- **findall(match, namespaces=None)**: Permite encontrar elementos específicos en el DOM.
- **find(match, namespaces=None)**: Retorna el primer subelemento que coincide con los criterios específicos.
- **findtext(match, default=None, namespaces=None)**: Retorna el texto que contiene un determinado nodo.

Veamos un ejemplo:

xml3.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <mediolocomocion>
3     <vehiculo>
4        <moto marca="bmw">Moto n&#250;mero 1</moto>
5        <auto marca="audi">Auto n&#250;mero 1</auto>
6        <moto marca="Yamaha">Moto n&#250;mero 1</moto>
7        <auto marca="Renault">Auto n&#250;mero 1</auto>
8     </vehiculo>
9  </mediolocomocion>
```

xml_elementtree3.py

```
1  import os
2  import xml.etree.ElementTree as ET
3  archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml3.xml"
4
5  import xml.etree.ElementTree as ET
6  dom = ET.parse(archivo)
7  raiz = dom.getroot()
8
9  # find the first 'item' object
```

```
10  for etiqueta in raiz:
11      print(etiqueta.find('auto').get('marca'))
12  print('---'*25)
13  for etiqueta in raiz:
14      for sub_etiqueta in etiqueta.findall('auto'):
15          print(sub_etiqueta.attrib)
16          print(sub_etiqueta.get('marca'))
```

Nos retorna:

```
audi
---------------------------------------------------------------------------
{'marca': 'audi'}
audi
{'marca': 'Renault'}
Renault
```

Modificar elementos

Para modificar el documento xml, contamos con varias herramientas, veamos ahora como cambiar el nombre de un nodo, cambiar el nombre de un atributo, modificar su valor, y agregar un atributo adicional a un elemento.

Notar que de la línea 10 a la 14 estamos accediendo al DOM y tratando a las etiquetas como si se tratara de listas, accediendo por posición, mientras que de la línea 17 en adelante accedemos a las etiquetas por su nombre mediante "**iter()**". En ambos casos para cambiar el contenido de texto de una etiqueta lo podemos hacer mediante "**.text**", y para agregar una atributo o modificarle su valor lo podemos hacer mediante "**set()**".

Para que los cambios sean persistidos se debe de finalizar guardando los cambios con "**write()**"

xmlelementtree4.py

```
1   import os
2   import xml.etree.ElementTree as ET
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml4.xml"
4
5   import xml.etree.ElementTree as ET
6   dom = ET.parse(archivo)
7   raiz = dom.getroot()
```

```
8
9     #Modificando una etiqueta por posición
10    print(raiz)
11    print(raiz[0])
12    print(raiz[0][0].text)
13    raiz[0][0].text = 'Moto modificada'
14    print(raiz[0][0].text)
15
16    #Modificando una etiqueta por nombre
17    for etiqueta in raiz.iter('auto'):
18        print(etiqueta.text)
19        etiqueta.text = 'Moto modificada por segunda vez'
20        print(etiqueta.text)
21
22    #Agregar un atributo
23    for etiqueta in raiz.iter('auto'):
24        etiqueta.set('color', 'rojo')
25
26    #Modificando un atributo
27    for etiqueta in raiz.iter('auto'):
28        etiqueta.set('color', 'azul')
29
30    dom.write(archivo)
```

Retorna:

```
−<mediolocomocion>
  −<vehiculo>
      <moto marca="bmw">Moto modificada</moto>
      <auto color="azul" marca="audi">Moto modificada por segunda vez</auto>
    </vehiculo>
  </mediolocomocion>
```

Crear sub elementos

Si necesitamos agregar elementos a un documento xml podemos utilizar:

- **makeelement ()**: Está función lleva como parámetros el nombre del nodo y un diccionario para incluir sus atributos.
- **SubElement ()**: En este caso debemos incluir el elemento principal y un diccionario de atributos como entradas.

Veamos un ejemplo:

xml_elementtree5.py

```
1   import os
2   import xml.etree.ElementTree as ET
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml4.xml"
4
5   import xml.etree.ElementTree as ET
6   dom = ET.parse(archivo)
7   raiz = dom.getroot()
8
9   # Agregamos elemento al elemento raíz
10  atributos = {}
11  print(raiz[0])
12  etiqueta = raiz.makeelement('vehiculos_aereos', atributos)
13  raiz.append(etiqueta)
14
15  # Agregamos elemento con atributo dentro del elemento creado en las líneas
    anteriores.
16  atributos2 = {'color': 'verde'}
17  avion = raiz[0][1].makeelement('avion', atributos2)
18  ET.SubElement(raiz[1], 'avion', atributos2)
19  raiz[1][0].text = 'Este es un avión'
20
21  dom.write(archivo)
```

Retorna:

```
−<mediolocomocion>
  −<vehiculo>
      <moto marca="bmw">Moto modificada</moto>
      <auto color="azul" marca="audi">Moto modificada por segunda vez</auto>
    </vehiculo>
  −<vehiculos_aereos>
      <avion color="verde">Este es un avión</avion>
    </vehiculos_aereos>
</mediolocomocion>
```

Borrar elemen”tos

Para eliminar elementos y atributos contamos con los métodos “remove()” y “pop()” respectivamente.

En el caso de “remove()” debemos pasarle como parámetro el nodo correspondiente, mientras que al utilizar “pop()” debemos pasar como parámetros el nombre del atributo y la opción “None” según podemos ver en el siguiente ejemplo en donde eliminamos el atributo “color” del auto y la etiqueta <moto>:

xml_elementtree6.py

```
1   import os
2   import xml.etree.ElementTree as ET
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml4.xml"
4
5   import xml.etree.ElementTree as ET
6   dom = ET.parse(archivo)
7   raiz = dom.getroot()
8
9   raiz[0][1].attrib.pop('color', None)
10
11  dom.write(archivo)
```

Retorna:

```
−<mediolocomocion>
  −<vehiculo>
      <auto marca="audi">Moto modificada por segunda vez</auto>
    </vehiculo>
  −<vehiculos_aereos>
      <avion color="verde">Este es un avión</avion>
    </vehiculos_aereos>
</mediolocomocion>
```

3.- minidom

Las distribuciones de Python traen también el módulo "**minidom**" con el cual trabajar sobre archivos xml, aún cuando la forma en la cual el documento xml es interpretado, difiere de cómo lo realiza "**ElementTree**".

Para analizar un documento XML utilizando minidom, primero debemos importarlo desde el módulo xml.dom y abrir el archivo utilizando la función parse():

```
from xml.dom import minidom

documento = minidom.parse(archivo)
```

Aquí el nombre del archivo puede ser una cadena que contiene la ruta del archivo o un objeto de tipo de archivo. La función devuelve un documento, que se puede manejar como un tipo XML. Por lo tanto, podemos usar la función getElementByTagName () para encontrar una etiqueta específica.

Dado que cada nodo puede tratarse como un objeto, podemos acceder a los atributos y al texto de un elemento utilizando las propiedades del objeto. En el siguiente ejemplo, hemos accedido a los atributos y al texto de un nodo específico, y de todos los nodos juntos de forma análoga a como lo realizamos con "ElementTree". Solo realizaremos la lectura de archivo a modo de ejemplo, se puede acceder a una referencia más completa desde el siguiente link:

https://docs.python.org/3.7/library/xml.dom.minidom.html

Leer archivo

xml_minidom1.xml

```
1   import os
2
3   from xml.dom import minidom
4   archivo = os.path.dirname(os.path.abspath(__file__))+"\\xml1.xml"
5
6   documento = minidom.parse(archivo)
7
8   varon = documento.getElementsByTagName('varon')
9   mujer = documento.getElementsByTagName('mujer')
10  print(varon[0].firstChild.data)
11  print(varon[0].childNodes[0].data)
12  print(varon[0].attributes['edad'].value)
13  print(mujer[0].firstChild.data)
14  print(mujer[0].childNodes[0].data)
15  print(mujer[0].attributes['edad'].value)
```

Nos retorna:

```
        Juan
        Juan
5
    Ana
    Ana
4
```

4.- JSON

El formato JSON también es un lenguaje de intercambios de datos como XML pero mucho más simple. Es un lenguaje muy utilizado en el envío de datos desde aplicaciones web y móviles realizadas con javascipt. Por su similitud con el tipo de datos de objetos de javascript a adquirido un papel muy importante en desarrolladores de esta comunidad. También es muy utilizado por los desarrolladores de videojuegos de y de animaciones para describir las partes del cuerpo de un avatar o darle formato a las texturas de un objeto.

El formato JSON para los que trabajamos con python sería muy similar asemejarlo a una lista de diccionarios, o sea que podemos pensar en la siguiente estructura:

```
[
  {
    " ":" ",
    " ":" ",
    " ":" ",

  },
    {
    " ":" ",
    " ":" ",
    " ":" ",
  },
]
```

De hecho cuando transformamos un json a python es entendido exactamente así, como una lista de diccionarios. Veamos un ejemplo en donde importamos un archivo ".json" y accedemos a uno de sus elementos, en el mismo si utilizamos **type()** podemos comprobar que estamos trabajando con listas y diccionarios.

Leer json

json1.json

```
1   {
2       "nombre": "Ana",
3       "edad": 33,
4       "estado_civil": true,
5       "esposo": "Pablo",
```

```
6       "hijos": ["Cecilia","Luis"],
7       "autos": [
8           {
9               "modelo": "Ford",
10              "color": "rojo"
11          },
12          {
13              "modelo": "Chevrolet",
14              "color": "azul"
15          }
16      ]
17  }
```

json1.py

```
1   import os
2   import json
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\json1.json"
4   archivo = json.loads(open(archivo).read())
5   print(archivo)
6   print("---"*25)
7   print(type(archivo))
8   print("---"*25)
9   print(type(archivo["autos"]))
10  print("---"*25)
11  print(type(archivo["autos"][0]))
12  print("---"*25)
13  print(archivo["autos"][0]["color"])
14  print("---"*25)
```

Retorna:

```
{'nombre': 'Ana', 'edad': 33, 'estado_civil': True, 'esposo': 'Pablo', 'hijos': ['Cecilia', 'Luis'],
'autos': [{'modelo': 'Ford', 'color': 'rojo'}, {'modelo': 'Chevrolet', 'color': 'azul'}]}
---------------------------------------------------------------------------
<class 'dict'>
---------------------------------------------------------------------------
<class 'list'>
---------------------------------------------------------------------------
<class 'dict'>
---------------------------------------------------------------------------
rojo
```

De python a json

Cuando pasamos de python a json, utilizamos el método “dump()” con el cual convertimos todos los tipos de objetos de python en objetos del tipo javascript, según la siguiente lista.

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	True
False	False
None	null

Para que el json quede mejor formateado podemos indicar la indentación, dado que una tabulación equivale a 4 espacios en el siguiente ejemplo elegimos una indentación de 4:

json2.py

```
1   import os
2   import json
3   archivo = os.path.dirname(os.path.abspath(__file__))+"\\json2.json"
4
5   data = {}
6   data['persona'] = []
7
8   data['persona'].append({
9       'nombre': 'Cecila',
10      'apellido': 'Rodriguez',
11      'age': 54,
12      })
13
14  data['persona'].append({
15      'nombre': 'Cecila',
16      'apellido': ['Rodriguez', 'Garcia'],
```

```
17      'age': 54,
18      })
19
20  with open(archivo, 'w') as file:
21      json.dump(data, file, indent=4)
```

json2.json

```
1   {
2       "persona": [
3           {
4               "nombre": "Cecila",
5               "apellido": "Rodriguez",
6               "age": 54
7           },
8           {
9               "nombre": "Cecila",
10              "apellido": [
11                  "Rodriguez",
12                  "Garcia"
13              ],
14              "age": 54
15          }
16      ]
17  }
```

Bibliografía utilizada y sugerida

Libros

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Programming Python 4th Edition – Mark Lutz – O'Reilly 2011

Manual online

https://docs.python.org/3.7/tutorial/

https://docs.python.org/3.7/library/index.html

https://www.w3.org/TR/REC-xml/

https://docs.python.org/3.7/library/xml.dom.minidom.html

https://docs.python.org/3.7/library/xml.etree.elementtree.html

Lo que vimos

En esta unidad hemos trabajado con los lenguajes xml y json de formateo de datos.

Lo que viene:

En la siguiente unidad veremos cómo trabajar con la librería socket..