

# Biblioteca *LXML*

Trabalhando com arquivos *XML* utilizando a biblioteca *LXML*

# XML e python-lxml

## O que é *XML*?

*XML* significa “*eXtensible Markup Language*.” ou “Linguagem de marcação extensível”

O *XML* foi projetado para armazenar e transportar dados.

Também foi projetado para ser legível para humanos e máquinas.

# XML e python-lxml

*XML* e *HTML* foram projetados com objetivos diferentes:

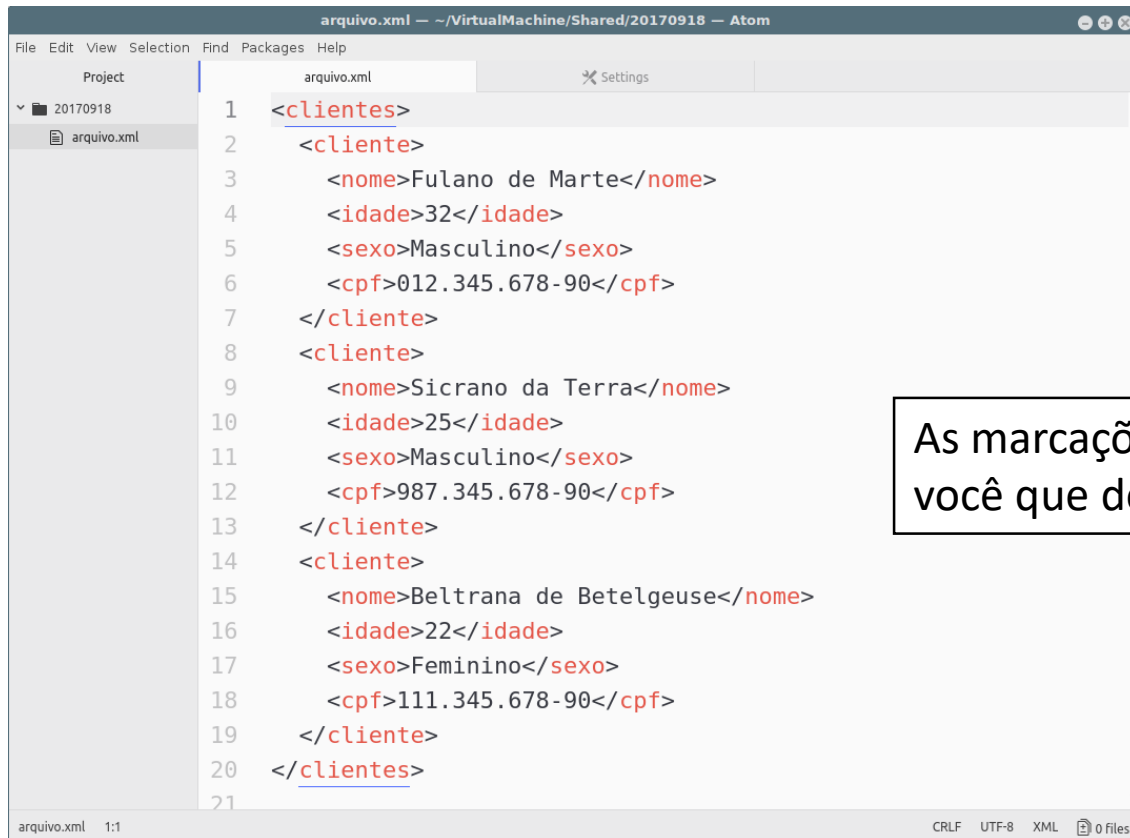
O *HTML* foi projetado para exibir dados - com foco em como os dados parecem.

O *XML* foi projetado para transportar dados – com foco no que o dado é.

As tags *XML* não são predefinidas, como as tags *HTML*.

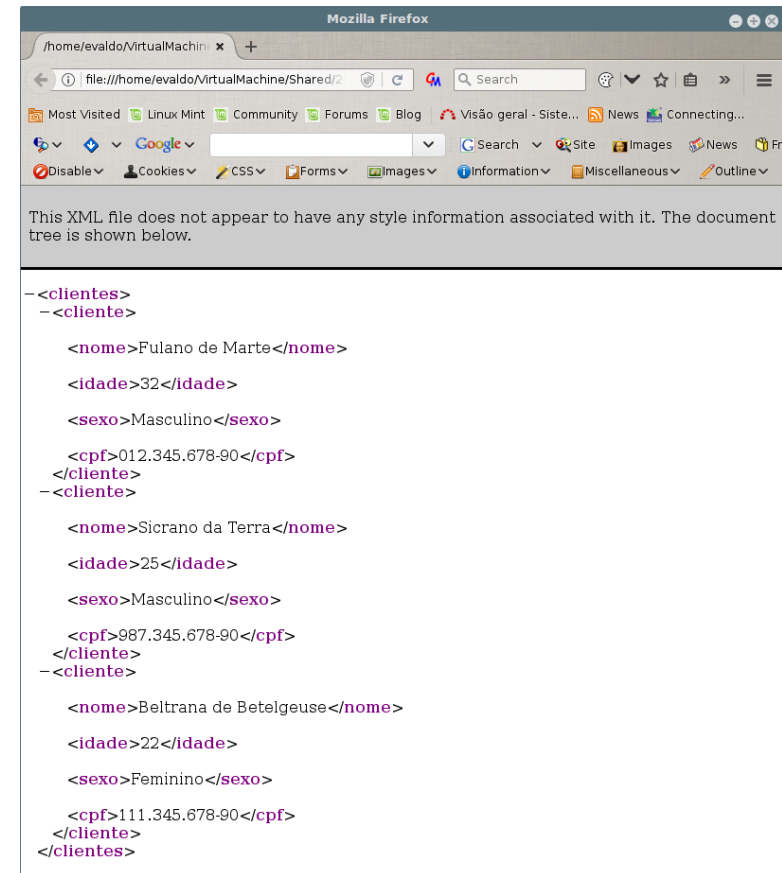
# XML e python-lxml

Veja um exemplo de um documento *XML*.



```
1 <clientes>
2   <cliente>
3     <nome>Fulano de Marte</nome>
4     <idade>32</idade>
5     <sexo>Masculino</sexo>
6     <cpf>012.345.678-90</cpf>
7   </cliente>
8   <cliente>
9     <nome>Sicrano da Terra</nome>
10    <idade>25</idade>
11    <sexo>Masculino</sexo>
12    <cpf>987.345.678-90</cpf>
13  </cliente>
14  <cliente>
15    <nome>Beltrana de Betelgeuse</nome>
16    <idade>22</idade>
17    <sexo>Feminino</sexo>
18    <cpf>111.345.678-90</cpf>
19  </cliente>
20 </clientes>
21
```

As marcações  
você que define.



```
--<clientes>
--<cliente>

  <nome>Fulano de Marte</nome>

  <idade>32</idade>

  <sexo>Masculino</sexo>

  <cpf>012.345.678-90</cpf>
</cliente>
--<cliente>

  <nome>Sicrano da Terra</nome>

  <idade>25</idade>

  <sexo>Masculino</sexo>

  <cpf>987.345.678-90</cpf>
</cliente>
--<cliente>

  <nome>Beltrana de Betelgeuse</nome>

  <idade>22</idade>

  <sexo>Feminino</sexo>

  <cpf>111.345.678-90</cpf>
</cliente>
</clientes>
```

# XML e python-lxml

## Modelo de Objeto de Documento (*DOM*)

O Modelo de Documento Objeto (do inglês *Document Object Model* - *DOM*) é uma convenção multiplataforma e independente de linguagem para representação e interação com objetos em documentos *HTML*, *XHTML* e *XML*. Os nós de cada documento são organizados em uma estrutura de árvore, chamada de árvore *DOM*. Os objetos na árvore *DOM* podem ser endereçados e manipulados pelo uso de métodos sobre os objetos.

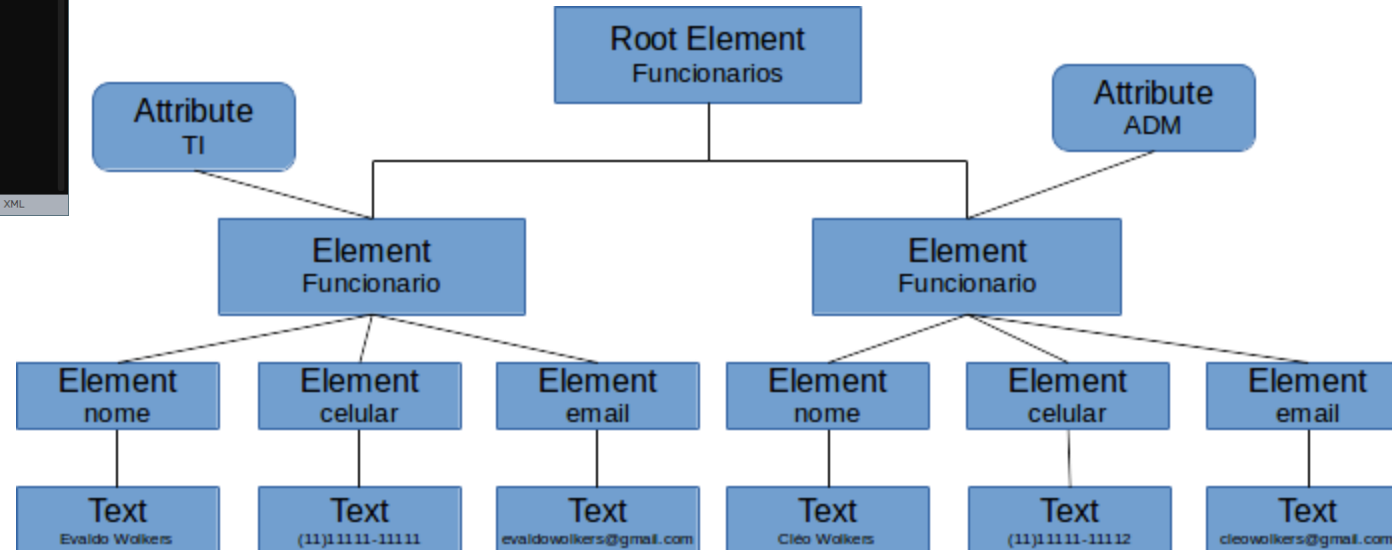
Fonte: [https://pt.wikipedia.org/wiki/Modelo\\_de\\_Objeto\\_de\\_Documentos](https://pt.wikipedia.org/wiki/Modelo_de_Objeto_de_Documentos)

# XML e python-lxml

## DOM

```
1 <?xml version = "1.0"?>
2 <funcionarios>
3   <funcionario departamento = "TI">
4     <nome>Evaldo Wolkers</nome>Texto após a tag nome
5     <celular>(11)11111-11111</celular>
6     <email>evaldowolkers@gmail.com</email>
7   </funcionario>
8
9   <funcionario departamento = "ADM">
10    <nome>Cleo Wolkers</nome>
11    <celular>(11)11111-11112</celular>
12    <email>cleowolkers@gmail.com</email>
13  </funcionario>
14 </funcionarios>
```

Esse texto fora da tag “nome”  
faz parte da tag mais externa  
“funcionario”.



# XML e python-lxml

## Analísadores *HTML* e *XML*

Até agora usamos o analisador *HTML* incluído na biblioteca padrão do Python:

```
bsObj = BeautifulSoup(html.read(), "html.parser")
```

O *HTML Parser* é a biblioteca de análise que vem embutida no *Python*. Não requer instalação e é extremamente fácil de usar.

Mas o *Python* também oferece suporte a vários analisadores de terceiros.

# XML e python-lxml

Nesta aula veremos o ***python-lxml***

Segundo a definição no próprio site do projeto (<http://lxml.de>), Lxml é a biblioteca mais funcional e fácil de usar para processar XML e HTML na linguagem Python.



# XML e python-lxml

## Instalando o *python-lxml*

```
Command Prompt
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Evaldo>pip install lxml
Collecting lxml
  Downloading lxml-3.8.0-cp36-cp36m-win_amd64.whl (3.2MB)
    100% |████████████████████| 3.2MB 119kB/s

Installing collected packages: lxml
Successfully installed lxml-3.8.0

C:\Users\Evaldo>
```

Formas de instalar, dependendo do seu ambiente.

```
$ apt-get install python-lxml
```

```
$ easy_install lxml
```

```
$ pip install lxml
```

```
evaldo@evaldo ~ $ sudo pip3.6 install lxml
The directory '/home/evaldo/.cache/pip/http' or its parent directory is not owned
by the current user and the cache has been disabled. Please check the permissi
ons and owner of that directory. If executing pip with sudo, you may want sudo's
-H flag.
The directory '/home/evaldo/.cache/pip' or its parent directory is not owned by
the current user and caching wheels has been disabled. check the permissions and
owner of that directory. If executing pip with sudo, you may want sudo's -H fla
g.
Collecting lxml
  Downloading lxml-4.0.0-cp36-cp36m-manylinux1_x86_64.whl (5.3MB)
    100% |████████████████████| 5.3MB 92kB/s
Installing collected packages: lxml
Successfully installed lxml-4.0.0
evaldo@evaldo ~ $
```

# XML e python-lxml

A biblioteca *lxml* é uma implementação *Python* para as bibliotecas *C libxml2* e *libxslt*. Ela é compatível, porém superior à *ElementTree* API. A classe *lxml.etree* da biblioteca *lxml* segue a *ElementTree* API o máximo possível.

Quando esta aula foi gravada a biblioteca estava em sua versão 4.0.0.

# XML e python-lxml

## ***lxml.etree***

O módulo `lxml.etree` implementa a API `ElementTree` estendida para XML.

Este é o link da documentação completa da API:

<http://lxml.de/4.0/api/index.html>

# XML e python-lxml

O primeiro passo é importar o módulo `lxml.etree`.

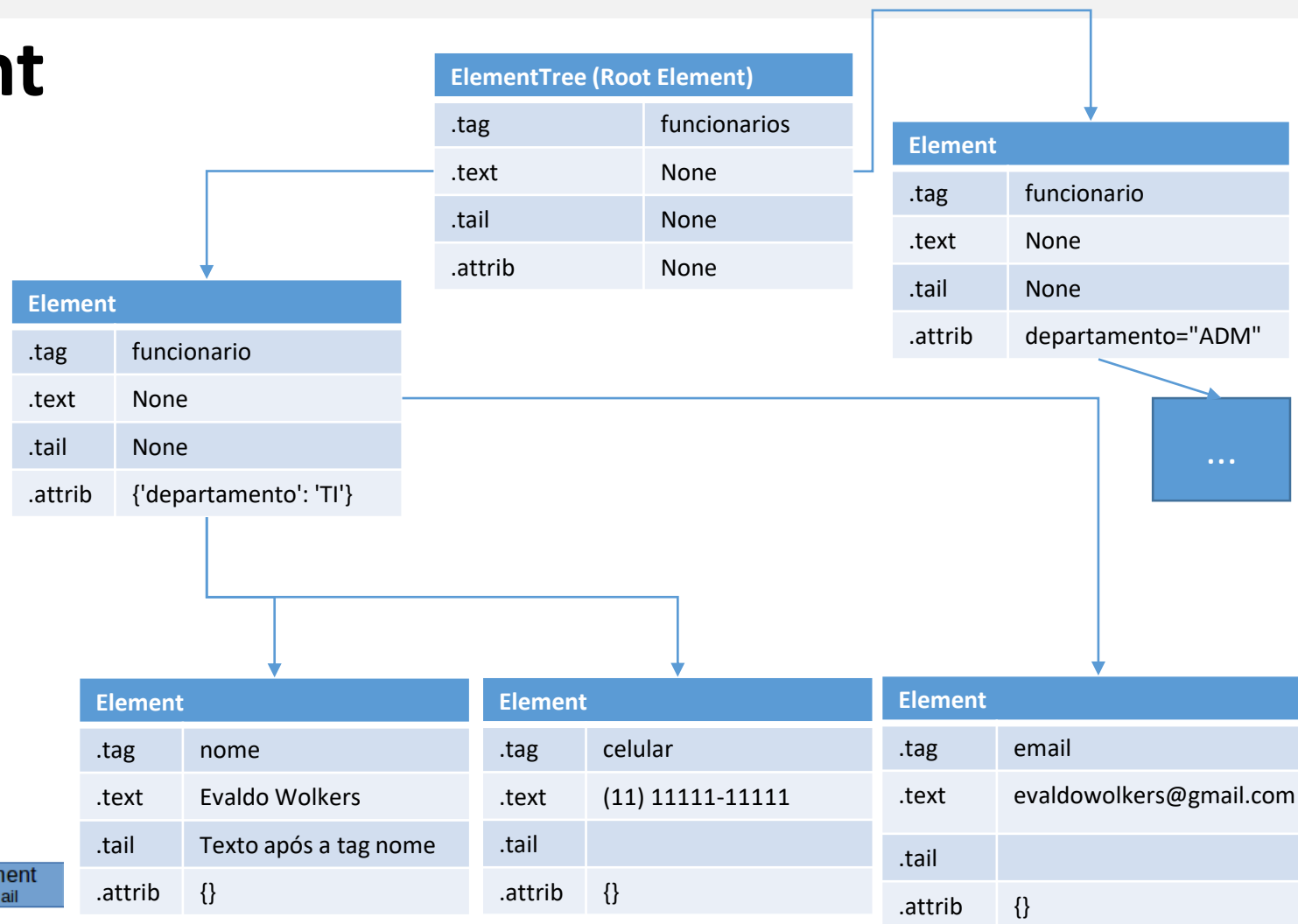
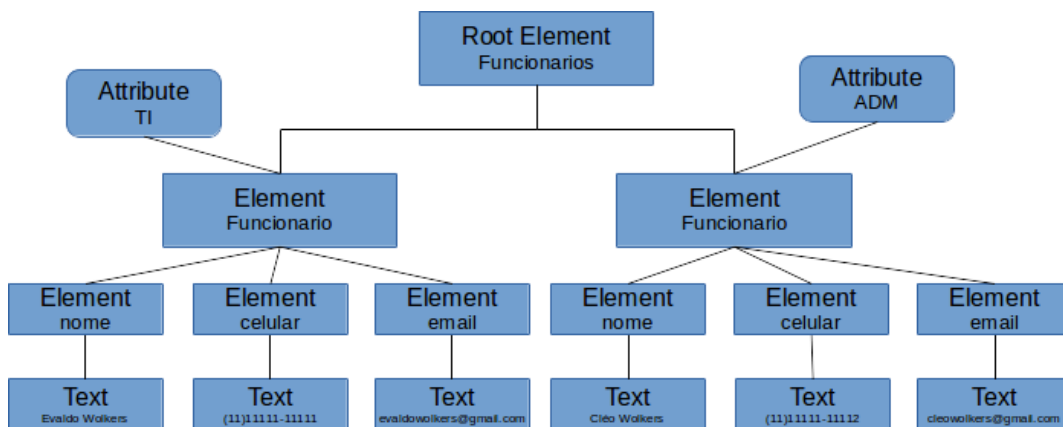
```
from lxml import etree
```

O módulo *etree* possui a classe *Element*. Um elemento é o objeto principal da *API ElementTree*. A maior parte das funcionalidades da árvore XML é acessada através desta classe. Podemos criar elementos facilmente utilizando *Element*.

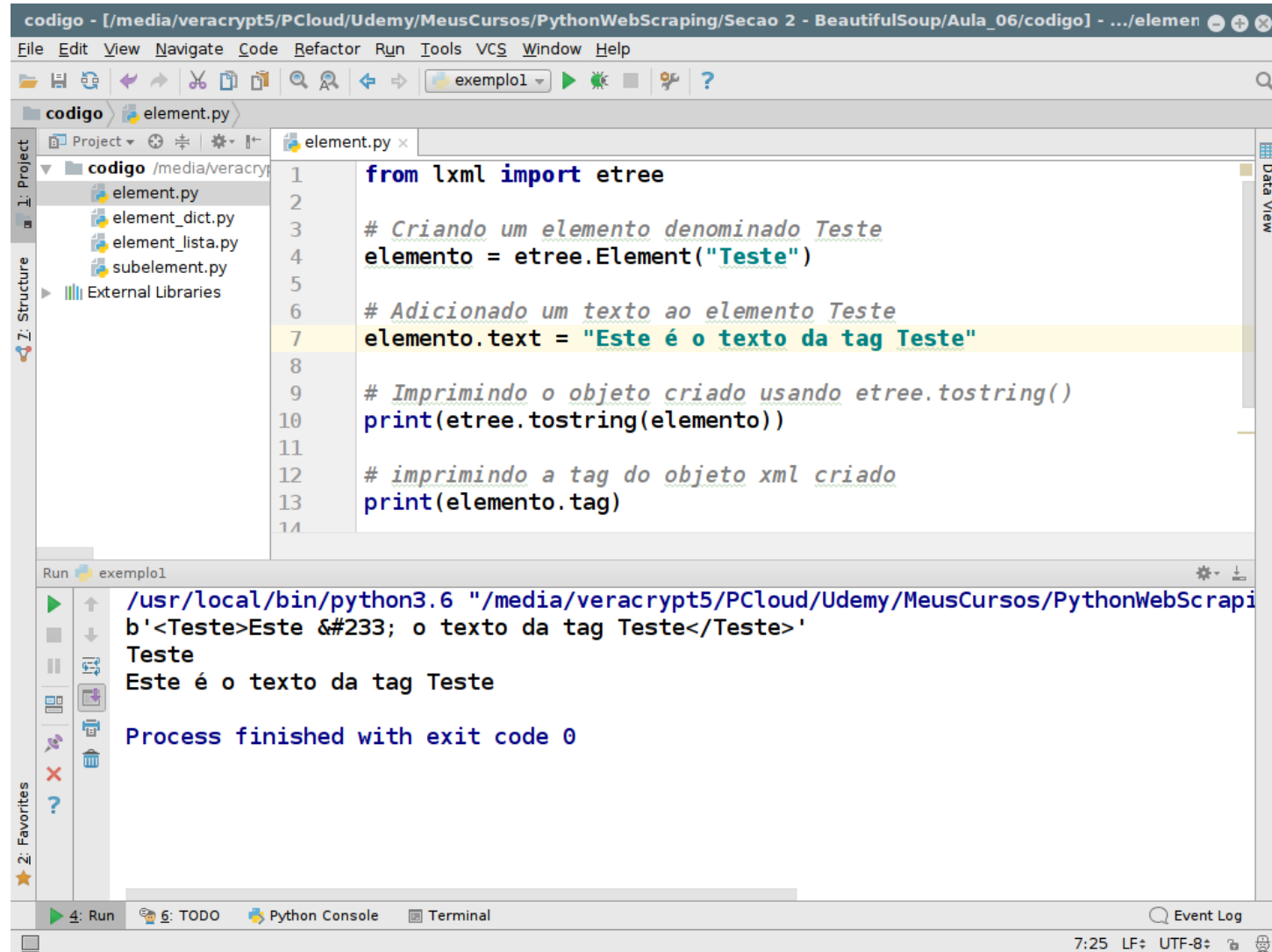
# XML e python-lxml

## Elementtree e Element

```
1 <?xml version = "1.0"?>
2 <funcionarios>
3   <funcionario departamento = "TI">
4     <nome>Evaldo Wolkers</nome>Texto após a tag nome
5     <celular>(11)11111-11111</celular>
6     <email>evaldowolkers@gmail.com</email>
7   </funcionario>
8
9   <funcionario departamento = "ADM">
10    <nome>Cleo Wolkers</nome>
11    <celular>(11)11111-11112</celular>
12    <email>cleowolkers@gmail.com</email>
13  </funcionario>
14 </funcionarios>
```



# XML e python-lxml



The screenshot shows a code editor window titled 'codigo - [/media/veracrypt5/PCloud/Udemy/MeusCursos/PythonWebScraping/Secao 2 - BeautifulSoup/Aula\_06/codigo] - .../elemen'. The editor contains a file named 'element.py' with the following Python code:

```
1 from lxml import etree
2
3 # Criando um elemento denominado Teste
4 elemento = etree.Element("Teste")
5
6 # Adicionado um texto ao elemento Teste
7 elemento.text = "Este é o texto da tag Teste"
8
9 # Imprimindo o objeto criado usando etree.tostring()
10 print(etree.tostring(elemento))
11
12 # imprimindo a tag do objeto xml criado
13 print(elemento.tag)
14
```

The code is executed, and the output is displayed in the 'Run' panel at the bottom. The output shows the XML string representation, the text content, and the tag name, followed by a confirmation message:

```
/usr/local/bin/python3.6 "/media/veracrypt5/PCloud/Udemy/MeusCursos/PythonWebScraping/Secao 2 - BeautifulSoup/Aula_06/codigo" - .../elemen
b'<Teste>Este é o texto da tag Teste</Teste>'
Teste
Este é o texto da tag Teste
Process finished with exit code 0
```

# XML e python-lxml

Para adicionar um subelemento podemos fazer de duas formas:

1) Criar o subelemento diretamente no *Element*:

```
clientes = etree.Element("clientes")  
cliente1 = etree.SubElement(clientes, "cliente")
```

2) Usar o método *append* de *Element*:

```
clientes = etree.Element("clientes")  
cliente1 = etree.Element("cliente")  
clientes.append(cliente1)
```

# XML e python-lxml

```
from lxml import etree

# Criando o elemento Clientes
clientes = etree.Element("clientes")

# Criando (diretamente) um subelemento do elemento clientes denominado clientel (tag cliente)
clientel = etree.SubElement(clientes, "cliente")
# Criando um subelemento do elemento clientel
# denominado nome1 (tag nome)
nome1 = etree.SubElement(clientel, "nome")
# Definindo um texto para nome1
nome1.text = "Fulano de Marte"
idade1 = etree.SubElement(clientel, "idade")
idade1.text = "32"
sexo1 = etree.SubElement(clientel, "sexo")
sexo1.text = "Masculino"
cpf1 = etree.SubElement(clientel, "cpf")
cpf1.text = "012.345.678-90"
```

```
"""
<clientes>
  <cliente>
    <nome>Fulano de Marte</nome>
    <idade>32</idade>
    <sexo>Masculino</sexo>
    <cpf>012.345.678-90</cpf>
  </cliente>
  <cliente>
    <nome>Sicrano da Terra</nome>
    <idade>25</idade>
    <sexo>Masculino</sexo>
    <cpf>987.345.678-90</cpf>
  </cliente>
  <cliente>
    <nome>Beltrana de Betelgeuse</nome>
    <idade>22</idade>
    <sexo>Feminino</sexo>
    <cpf>111.345.678-90</cpf>
  </cliente>
</clientes>
"""
```

```
cliente2 = etree.Element("cliente")
nome2 = etree.SubElement(cliente2, "nome")
nome2.text = "Sicrano da Terra"
# Adicionando com append o elemento cliente2 no elemento
# clientes (vai ser um subelemento)
idade2 = etree.SubElement(cliente2, "idade")
idade2.text = "25"
sexo2 = etree.SubElement(cliente2, "sexo")
sexo2.text = "Masculino"
cpf2 = etree.SubElement(cliente2, "cpf")
cpf2.text = "987.345.678-90"
clientes.append(cliente2)
```

```
cliente3 = etree.Element("cliente")
nome3 = etree.SubElement(cliente3, "nome")
nome3.text = "Beltrana de Betelgeuse"
# Adicionando com append o elemento cliente2 no elemento clientes (vai ser um subelemento)
idade3 = etree.SubElement(cliente3, "idade")
idade3.text = "22"
sexo3 = etree.SubElement(cliente3, "sexo")
sexo3.text = "Feminino"
cpf3 = etree.SubElement(cliente3, "cpf")
cpf3.text = "111.345.678-90"
clientes.append(cliente3)

print(etree.tostring(clientes))
print(etree.tostring(clientes, pretty_print=True))
print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))
```



# XML e python-lxml

## ***Elements* são listas**

Para facilitar o acesso aos subelementos, um *Element* imita o comportamento de listas do Python.

É possível manipular o objeto *Element* como se fosse uma lista e também converter em lista com *list()*.

# XML e python-lxml

```
from lxml import etree

clientes = etree.Element("clientes")
cliente1 = etree.SubElement(clientes, "cliente1")
nome1 = etree.SubElement(cliente1, "nome")
nome1.text = "Fulano de Marte"

cliente2 = etree.Element("cliente2")
nome2 = etree.SubElement(cliente2, "nome")
nome2.text = "Anabelle de Saturno"
clientes.append(cliente2)

cliente3 = etree.Element("cliente3")
nome3 = etree.SubElement(cliente3, "nome")
nome3.text = "Plutonilda"
clientes.append(cliente3)

# len(Element) retorna a quantidade de tags do objeto
print("Total de clientes:", len(clientes))

# Criando um objeto com base no segundo elemento
cliente_dois = clientes[1]
# Imprimindo a tag do elemento de índice 1, que é
print("Tag clientes[1]:", cliente_dois.tag)
```

```
# Percorrendo o Element como uma lista
for x in clientes:
    print(x.tag)

print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))

# Inserindo um elemento usando insert
clientes.insert(0, etree.Element("cliente0"))
print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))

# Fatando a lista clientes, pegando os elementos de 1 a 2
# porque [1:3] não inclui o 3
fatial = clientes[1:3]
for x in fatial:
    print(x.tag)

print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))
# O elemento da posição 2 (cliente2) vai ser substituído
# pelo elemento da posição 1 cliente1, ficando apenas os
# elementos 0, 1 e 3
clientes[2] = clientes[1]
print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))

# Verifica se clientes é 'pai' de clientes[1]
print(clientes is clientes[1].getparent())
```

# XML e python-lxml

## Atributos de elementos XML (Dicionários)

Elementos XML aceitam atributos, veja um exemplo:

```
<pessoa sexo='masculino'>
```

*Element* aceita adicionar atributos utilizando dicionários (conjunto “chave=valor”).

# XML e python-lxml

```
from lxml import etree

# Como elementos XML aceitam atributos
# Você pode adicionar atributos como dicionários chave=valor
clientes = etree.Element("clientes", atributo = "valor")
cliente1 = etree.SubElement(clientes, "cliente1")
nome1 = etree.SubElement(cliente1, "nome")
nome1.text = "Fulano de Tal"

print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))

# Adicionando um atributo denominado código com valor 1248
clientes.set("codigo", "1248")
print(etree.tostring(clientes, pretty_print=True).decode("utf-8"))
# Imprimindo o valor do atributo código
print(clientes.get("codigo"))

# Imprimindo os atributos de clientes (Element)
print(clientes.keys())
```

```
for atributo, valor in sorted(clientes.items()):
    print(f"{atributo} = {valor}")

# Pegando os atributos
atributos = clientes.attrib

print(atributos["codigo"])
print(atributos.get("atributo-inexistente"))

atributos["atributo"] = "Valor do atributo"
print(atributos["atributo"])
print(clientes.get("atributo"))

# Atributos podem ser copiados para dicionários
d = dict(clientes.attrib)
print(sorted(d.items()))
```

# XML e python-lxml

## Lendo arquivos *XML* com *lxml*

```
from lxml import etree

# funcionários será do tipo ElementTree com vários elementos
funcionarios = etree.parse("funcionarios.xml")

# Localizando a primeira ocorrência da tag funcionario
print(funcionarios.find("funcionario"))

# Localizando a primeira ocorrência da tag funcionario
print(funcionarios.getroot().find('funcionario'))

# Localizando todas ocorrências da tag funcionario
print(funcionarios.findall("funcionario"))

# Imprimindo o documento XML com getroot
print(etree.tostring(funcionarios.getroot(), pretty_print=True).decode("utf-8"))

# Copiando os Elements existentes no ElementTree
todos_funcionarios = funcionarios.findall("funcionario")
```

```
# Imprimindo quantos Elements encontrados
#print(len(todos_funcionarios))

for funcionario in todos_funcionarios:
    print("=====")
    print("Tag:", funcionario.tag.strip())
    print("Text:", funcionario.text.strip())
    print("Tail:", funcionario.tail.strip())
    print("Attrib:", funcionario.attrib)

    for informacao in funcionario:
        print("*****")
        print("Tag:", informacao.tag.strip())
        print("Text:", informacao.text.strip())
        print("Tail:", informacao.tail.strip())
        print("Attrib:", informacao.attrib)
        print("*****")
    print("=====")

"""
Tag: Nome da Tag
Text: Texto da Tag
Tail: Texto após a Tag
Attrib: Atributos
"""
```

```
<?xml version = "1.0"?>
<funcionarios>
  <funcionario departamento = "TI">
    <nome>Evaldo Wolkers</nome>Texto após a tag nome
    <celular>(11)11111-11111</celular>
    <email>evaldowolkers@gmail.com</email>
  </funcionario>
  <funcionario departamento = "ADM">
    <nome>Cleo Wolkers</nome>
    <celular>(11)11111-11112</celular>
    <email>cleowolkers@gmail.com</email>
  </funcionario>
</funcionarios>
```

# FIM