Лабораторно упражнение №6

Използване на уеб услуги в Python, обработване на XML структури, обработване на JSON структури, използване на приложни програмни интерфейси (API)

Има два често използвани формата, които се използват при обмен на данни в мрежата. Форматът eXtensible Markup Language (XML) се използва от много дълго време и е най-подходящ за обмен на данни в стил на документ. Когато програмите просто искат да обменят масиви, списъци или друга вътрешна информация помежду си, те използват JavaScript Object Notation (JSON) (вижте www.json.org). В това упражнение ще разгледаме и двата формата.

Основни понятия:

- API (Application Program Interface) - връзка между приложения, която определя моделите на взаимодействие между два компонента на приложението.
- ElementTree - Вградена библиотека на Python, използвана за анализ на XML данни.
- JSON (JavaScript Object Notation) - формат, който позволява маркиране на структурирани данни въз основа на синтаксиса на JavaScript обекти.
- SOA (Service-Oriented Architecture) – използва се при приложения, свързани в мрежа.
- XML (eXtensible Markup Language) - формат, който позволява маркиране на структурирани данни.

1. eXtensible Markup Language - XML

XML изглежда много подобно на HTML, но е по-структуриран от HTML. Ето пример за XML документ:

```
<person>
        <name>Chuck</name>
        <phone type="intl">
                +359 822456789
        </phone>
        <email hide="yes" />
</person>
```

Всяка двойка отварящи (например <person>) и затварящи тагове (например, <\ person>) представлява елемент или възел със същото име като маркера (например person). Всеки елемент може да има някакъв текст, някои атрибути и други вложени елементи. Ако XML елементът е празен (т.е. няма съдържание), той може да бъде изобразен чрез самозатварящ се маркер (например, <email />).

2. XML

Ето едно просто приложение, което анализира и извлича някои елементи от XML:

```
import xml.etree.ElementTree as ET
data = '''
<person>
      <name>Chuck</name>
      <phone type="intl">
            +1 734 303 4456
      </phone>
      <email hide="yes" />
</person>'''
tree = ET.fromstring(data)
print('Name:', tree.find('name').text)
print('Attr:', tree.find('email').get('hide'))
```

Тройният единичен цитат (''' '''), както и тройният двоен цитат (""" """), позволяват създаването на низове, които обхващат няколко реда.

Когато XML е със структура на дърво, имаме серия от методи, които могат да се използват за извличане на данни от XML низ. Функцията find търси през XML дървото и извлича елемента, който съответства на указания маркер.

```
Name: Chuck
Attr: yes
```

Използването на ElementTree има предимството, че можем да извличаме данни от XML, без да се притесняваме за правилата на XML синтаксиса.

3. Обработка на възли

Често XML има множество възли и за обработката им се използва оператор за цикъл. В следната програма се преглеждат всички потребителски възли:

```
import xml.etree.ElementTree as ET
input = '''
<stuff>
<users>
<user x="2">
<id>001</id>
<name>Chuck</name>
</user>
<user x="7">
<id>009</id>
<name>Brent</name>
</user>
</users>
</stuff>'''
stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
```

```
print('User count:', len(lst))
for item in lst:
      print('Name', item.find('name').text)
print('Id', item.find('id').text)
      print('Attribute', item.get('x'))
```

Методът findall извлича списък на подредове, които представляват потребителските структури в XML дървото.

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

Важно е да се включат всички елементи на ниво родител в оператора findall с изключение на елемента от най-високо ниво (например, users/user). В противен случай Python няма да намери желаните възли.

```
import xml.etree.ElementTree as ET
input = '''
<stuff>
<users>
<user x="2">
<id>001</id>
<name>Chuck</name>
</user>
<user x="7">
<id>009</id>
<name>Brent</name>
</user>
</users>
</stuff>'''
stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))
lst2 = stuff.findall('user')
print('User count:', len(lst2))
```

lst съхранява всички потребителски елементи, които са вложени в родителя. **lst2** търси потребителски елементи, които не са вложени в елемента от най-високо ниво.

```
User count: 2
User count: 0
```

4. JavaScript Object Notation- JSON

Форматът JSON е вдъхновен от формата на обекти и масиви, използвани в езика на JavaScript. Но тъй като Python е изобретен преди JavaScript, синтаксисът на Python за масиви и списъци повлиява на синтаксиса на JSON. В този смисъл форматът на JSON е идентичен с този на Python.

Ето JSON, който е приблизително еквивалентен на простия XML по-горе:

```
{
"name" : "Chuck",
"phone" : {
"type" : "intl",
"number" : "+1 734 303 4456"

},
"email" : {
"hide" : "yes"
}
}
```

Забелязват се някои разлики. Първо, в XML можем да добавим атрибути като „intl“ към етикета „phone“. Данните в JSON се състоят от двойки <ключ-стойност>. Също така XML етикетът „person“ липсва, заменен от набор от външни фигурни скоби.

Като цяло JSON структурите са по-прости от XML, защото JSON има по-малко възможности от XML. Предимството на JSON формата е, че се прехвърля директно към някаква комбинация от списъци.

Методът json.loads(<json_data>) връща списък с данните на структура json_data.

Самостоятелна задача 1:

Да се напише програма на Python, която връща списък на потребители, като всеки потребител е набор от двойката <ключ-стойност>.

Самостоятелна задача 2:

Дадена е следната JSON структура:

{

"name": "Bob",

"languages": ["English", "Fench"]

}

Да се напише програма на Python, която преобразува дадената структура в Python речник (dictionary).

Самостоятелна задача 3:

Да се създаде файл, съдържащ следните JSON обекти.

Person.json

{"name": "Bob",

```
"languages": ["English", "Fench"]
}
```

Да се напише програма на Python, която прочита файла и извежда съдържанието му.

Самостоятелна задача 4:

Дадена е следната речникова структура:

```
{
'name': 'Bob',
'age': 12,
'children': None
}
```

Да се напише програма на Python, която конвертира дадената речникова структура в JSON обект.

Самостоятелна задача 5:

Дадена е следната JSON структура:

```
{
"name": "Bob",
"languages": ["English", "Fench"],
"married": True,
"age": 32
}
```

Да се напише програма на Python, която записва дадената JSON структура във файл.

Самостоятелна задача 6:

Да се напише програма, която позволява анализиране и отстраняване на грешки от JSON данни чрез отпечатване на данните в разбираем вид.

Полезни връзки.

1. https://docs.python.org/3/
2. https://www.programiz.com/python-programming
3. https://www.tutorialspoint.com/python/index.htm
4. https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ch1.html