Json

JSON(Javascript Object Notation) - невеликий за обсягом формат обміну даних, зроблений за зразком синтаксису JavaScript. У мові програмування Python модуль json має деякий функціонал для роботи з json файлами. Функція dumps кодує об’єкти з мови Python у об’єкти формату json.(як-от: list, dictionary тощо)

Приклад використання:

>>> import json

>>> json.dumps([1,2,3,{'4': 5, '6': 7}], separators=(',', ':'))

'[1,2,3,{"4":5,"6":7}]'

Також, функція loads та функція load перетворюють json на об’єкти Python: array на list, object на dictionary, працюють з даними форматів string, int, float та перетворює true та false на True та False відповідно.

Приклад використання:

>>> import json

>>> json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]')

['foo', {'bar': ['baz', None, 1.0, 2]}]

Також реалізована можливість створювати енкодери та декодери, параметри яких можна кастомізовувати.

html.parser

Цей модуль визначає клас HTMLParser який дозволяє базовий парсинг текстових файлів у HTML та XHTML форматуванні, навіть за умови неправильної розмітки останніх.

Метод HTMLParser.feed(data) подає деякий текст для парсингу, парсить лише завершені елементи, незавершені елементи буферизуються, поки не будуть подані нові дані або не буде викликаний метод close().

Приклад використання:

>>> parser.feed('<h1>Python</h1>')

Start tag: h1

Data : Python

End tag : h1

Метод close() примусово обробляє всі буферизовані файли так, ніби файл вже скінчився.

Метод reset() видаляє всі необроблені дані та скидає екземпляр.

Метод getpos() повертає номер поточного рядка та зміщення.

Метод HTMLParser.get\_starttag\_text() повертає текст останнього відкритого тегу.

Приклад парсингу невалідного html:

>>> parser.feed('<p><a class=link href=#main>tag soup</p ></a>')

Start tag: p

Start tag: a

attr: ('class', 'link')

attr: ('href', '#main')

Data : tag soup

End tag : p

End tag : a

XML

Модуль xml.etree.ElementTree використовує простий та ефективний API для того, щоб парсити та створювати дані у форматі XML.

XML- ієрархічний формат даних, тому ці дані простіше представляти у вигляді дерев. Тому є 2 класи: ElementTree, який представляє весь XML у вигляді дерева та клас Element, який представляє вершини дерева. Ці дані можуть бути зчитані як з файла, так і з текстової стрічки.

Метод Element.findall(tag) знаходить всі елементи, які є прямими нащадками цього елемента та мають цей тег.

Метод Element.find(tag) знаходить перший елемент-нащадок, який має цей тег.

Метод Element.text отримує доступ до текстових даних елемента.

Метод Element.clear() повністю очищує елемент.

Метод Element.insert додає піделемент у елемент на місце з заданим індексом.

Метод Element.remove видаляє піделемент з елементу

У класі ElementTree методи find та findall працюють так само, як і в класі Element.

Приклад роботи модуля:

<?xml version="1.0"?>

<data>

<country name="Liechtenstein">

<rank>1</rank>

<year>2008</year>

<gdppc>141100</gdppc>

<neighbor name="Austria" direction="E"/>

<neighbor name="Switzerland" direction="W"/>

</country>

<country name="Singapore">

<rank>4</rank>

<year>2011</year>

<gdppc>59900</gdppc>

<neighbor name="Malaysia" direction="N"/>

</country>

<country name="Panama">

<rank>68</rank>

<year>2011</year>

<gdppc>13600</gdppc>

<neighbor name="Costa Rica" direction="W"/>

<neighbor name="Colombia" direction="E"/>

</country>

</data>

import xml.etree.ElementTree as ET

tree = ET.parse('country\_data.xml')

root = tree.getroot()

>>> for child in root:

... print(child.tag, child.attrib)

...

country {'name': 'Liechtenstein'}

country {'name': 'Singapore'}

country {'name': 'Panama'}