**JSON encoder for Python (json)**

For working with json format in python, library with the same name ("json") can be used. There are different functions in it. In example, json.dump(...) is used to create custom json stream, which can be written into a file or output as string. By default, dict keys that are not of a basic type (str, int, float, bool, None) are skipped. Also, it ignores non-ASCII characters by default. By default, allow\_nan is True. It allows to use nan, inf, and -inf by turning them into NaN, Infinity, -Infinity when in json. If it is False, ValueError is raised when attempting to use inf, nan, or -inf. Programmer can change separators for changing string representation of json. This can be used for more easy to use or compact representation. sort\_keys parameter can be changed to True so that resulted dictionaries will be sorted by keys. You can't use multiple dump() to serialize multiple objects. One more thing. In json, keys are always string. That's why turning dictionary into json and back will change the dictionary.

There is also json.dumps(...). It is the same as json.dump, but it is used only when you want to get string.

In order to decode json information, method json.joads(...) can be used. It takes string representation of a JSON information and returns information in forms of lists and dictionaries. Other method for decoding is json.load(...), but this one is for reading files instead of strings.

There is a way to encode custom variables like complex numbers using JSON. For this, you have to specify object\_hook parameter of json.loads(...). This parameter must be a function, in which it is specified how values must be represented.

It is also possible to extend json.JSONEncoder class in order to encode custom objects like complex numbers.

**HTML and XHTML parser for Python (html.parser)**

This module defines class HTMLParser, which can be used to parse text files formatted in HTML. It has separate methods for handling start tag, end tag, and data. Each of those is intuitively easy to use. You can write anything in those methods. They will be called each time the program reads given part of the document. Other similar "handlers" are present, like handle\_comment(data), handle\_charref(name), and others.

Other methods are: HTMLParser.feed(data) for "feeding" string data into a parser, HTMLParser.close() for stopping parser at any point (can be used for some additional processing after parsing), HTMLParser.reset() for resetting the instance and losing all unprocessed data. HTMLParser.getpos() returns line number and offset.

In short, it is a very easy to use module, but because of this fact I can't write much about it.

**XML Processing Modules (packed in the 'xml' package)**

This pack of modules has different ways of processing xml files. In example, xml.etree.ElementTree, xml.dom, xml.dom.minidom, xml.dom.pulldom. I don't want to go into too much details about those because I won't use them anyway – module for parsing information will do everything for me.

I will start with the fact that these modules aren't safe against maliciously constructed data. More than half of stuff is vulnerable.

I will start from describing xml.etree.ElementTree. It has method parse(), which returns document as an object which can be read using different methods of the object (for example, getroot()). Object is structured as tree of elements. For example, you get root using object.getroot(). It has some data and collection of children. Obviously, children can have other children.

By the way, this module skips some unnecessary elements like comments, processing instructions. But these will be included when generating XML output.

There is a method ElementTree.XMLPullParser(). It allows to parse xml file event by event. This can be used, in example, if information is read part by part from a database.

All methods of each element of children are intuitively understandable as parts of a tree.

This module can be used for creating your own xml files. Or modifying existing ones. This can be used by method tree.write().

When creating a tree, elements of it can be defined as variables. And then you can create children for those elements using SubElement method of ElementTree. By the way, new elements are created using ElementTree.Element.

There is also a method for finding all elements with given prefixes.

The module offers some support for something called XPath expressions for locating elements in a tree.

Now about different methods. You can create comments, dump elements one by one, parse xml as string, check if element is element of a tree, and other useful ones for that task.

DOM is a cross-language thing for accessing xml data. File is rendered as tree of objects called nodes with siblings and children. There is also different type of node called DocumentType, which has some useful attributes for a document like id, name, notations, and etc. Entire XML document is represented by class Document. Element is a subclass of Node and inherits all its attributes. Attributes are attr objects. There are also comment and processing instruction comments. And others.

**lxml – library for processing XML and HTML**

Element tree here is pretty similar to that of a previous module. Elements can be defined as objects, special method can create sub-elements. It is also possible to parse files, obviously. I won't write anything more about this one, because I believe it shares some functionality with previously described modules.