**Description of JSON opportunities**

***JSON*** (or *Java Script Object Notation*) – is a text-format data transmission between computers. JSON is based on text that can be readable by human. Format allows describing objects and other data structures.

This format is usually used to transmit structured information in the Internet, based on process called *serialization*.

JSON was developed by Douglas Crockford.

JSON is used in programs developing with using technology called AJAX.

JSON, which is used in AJAX, usually appears as substitute of XML in the time of asynchronous transmission of structured data between the client and the server. When we compare JSON and XML, the main advantage of JSON is that it allows difficult structures in attributes, takes less space and interpret into object with Java Script.

Usage of JSON

Because JSON is more laconic than XML, it is more suitable for serialization of difficult structures.

When we talk about web applications, it that key it is relevant:

1) in problems of data transmission between browser and server;

2) in problems of data transmission between servers (HTTP interfaces).

JSON is a good choice for storage of difficult dynamical structures in relational bases or file cash.

Example of JSON object

JSON builds on two structures:

* A set of pairs name/value. In languages, it is realized like an object, a notation, a structure, a vocabulary, a hash-table, a list with keys or an associational array.
* An ordered list of values. In languages, it is realized like an array, a vector, a list, or a sequence.

This universal data structures. Theoretically, all modern languages use it in this or another form. As JSON is used for data transmission between different programming languages it makes sense to build it on these structures.

In JSON we use these kinds of their forms:

* **Object** – a sequence of name/value. Object starts with “{“ and ends with “}”. Every value is written after “:” and pairs name/value are separated with “,”.
* **Array** – a sequence of values. Array starts with “[“ and ends with “]”. Values are separated with “,”.
* **Value** can be a string in “ ”, or a number, or logical *true/false*, or null, or an object, or an array. These structures can contain each other.
* **String** – a sequence, which starts from 0 or more Unicode symbols, limited by “ ”, with usage of escape-sequences which start from backslash. Symbols are submitted as simple string.

Type **String** looks very similar like String in C or Java. Number is also very similar to C-number or Java-number, except that octal and hexadecimal formats are not used.

This example shows JSON object which describe a person.

It contains string places for name and surname of a person, an object which describes an address and an array which contains the list of phone numbers.

{

"firstName": "Ivan",

"lastName": "Kovalenko",

"address": {

"streetAddress": "Grushevskogo str, 14, flat 101",

"city": "Kyiv",

"postalCode": 21000

},

"phoneNumbers": [

"044 123-1234",

"050 123-4567"

]

}

Python & JSON

Python contains a library for working with JSON. There are two special modules:

1) marshal - convert Python objects to streams of bytes and back (with different constrains)

2) pickle - convert Python objects to streams of bytes and back.

There are also several methods, which we can use to work with JSON:

* json.**dump**(obj, fp, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw) - serialize object as a JSON formatted stream to fp (a .write()-supporting [file-like object](https://docs.python.org/3.5/glossary.html" \l "term-file-like-object)) using this [conversion table](https://docs.python.org/3.5/library/json.html" \l "py-to-json-table).
* json.**dumps**(*obj*, *skipkeys=False*, *ensure\_ascii=True*, *check\_circular=True*, *allow\_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort\_keys=False*, *\*\*kw*) - serialize object to a JSON formatted [str](https://docs.python.org/3.5/library/stdtypes.html#str)ing using this [conversion table](https://docs.python.org/3.5/library/json.html" \l "py-to-json-table). The arguments have the same meaning as in [dump()](https://docs.python.org/3.5/library/json.html" \l "json.dump" \o "json.dump).
* json.**load**(*fp*, *cls=None*, *object\_hook=None*, *parse\_float=None*, *parse\_int=None*, *parse\_constant=None*, *object\_pairs\_hook=None*, *\*\*kw*) - deserialize *fp* (a .read()-supporting [file-like object](https://docs.python.org/3.5/glossary.html" \l "term-file-like-object) containing a JSON document) to a Python object using this [conversion table](https://docs.python.org/3.5/library/json.html" \l "json-to-py-table).
* json.**loads**(*s*, *encoding=None*, *cls=None*, *object\_hook=None*, *parse\_float=None*, *parse\_int=None*, *parse\_constant=None*, *object\_pairs\_hook=None*, *\*\*kw*) - deserialize *s* (a [str](https://docs.python.org/3.5/library/stdtypes.html#str) instance containing a JSON document) to a Python object using this [conversion table](https://docs.python.org/3.5/library/json.html" \l "json-to-py-table).

References:

# 1) <https://docs.python.org/3.5/library/json.html> - [json](https://docs.python.org/3.5/library/json.html#module-json) — JSON encoder and decoder.

2) <http://www.json.org/>