Cal Poly **CSC 369:** Distributed Computations Alexander Dekhtyar

JSON

JSON

JSON, short for JavaScript Object Notation is a human- and machine-readable serialization mechanism for representing collections of key-value pairs[1].

Properties. JSON has the following nice properties.

- **JSON is plain text.** JSON objects are plain text objects that can be viewed and read by humans.
- **JSON is lightweight.** JSON specification is very simple.
- **JSON is structured.** JSON objects can contain other JSON objects in them allowing for structured data representation.
- **JSON is schemaless.** JSON does not require a schema to operate. This means JSON objects can be used to conveniently represent semi-structured data.

JSON Specification

JSON objects can be specified formally (in Backus-Naur notation) as follows:

```
<JSON Object> ::=  '{' '}' |
                   '{' <string> ':' <value>  (',' <string> ':' <value>)* '}'

<JSON Array> ::= '[' ']' |
                 '['  <value> (',' <value>)* ']'
```

[1] http://www.json.org

```
<value> :: =  <string> | <number>  |
              <JSON object> | <JSON array> |
              true | false | null
```

Here:

- Identifiers in angle brackets (e.g., <JSON Object> or <string>) otherwise called *non-terminals* are specific parts of the described syntax that are being defined.
- The ::= symbol is the "is defined as" notation.
- Items in single quotes (e.g., '{' or ']') are *terminals* or the actual symbols used in the JSON syntax.
- The || is the "or" symbol stating that a specific notion can be defined in more than one way.
- The (...)* notation means *one or more copy of what is inside the parenthises.*

With this in mind, here is a translation:

- A JSON Object is either an empty object { } or a collection of comma-separated key-value pairs, inside curly braces, where the key is a string object, and the value is a value object.
- A JSON Array is either an empty array [ ] or a sequence of comma-separated value objects inside angle brackets.
- A value object in JSON is either a single string object, or a single number object, or a single JSON object or a single JSON array. In addition, three trivial value objects exist: true, false and null.
- string objects are sequences of characters in quotes. number objects follow the standard syntax for numeric notation for either *integer* or *floating point* numbers. This includes scientific notation.

Examples. Here are some sample JSON objects.

```
{ "name": "Bob",
  "class": "senior",
  "grades": ["A", "A", "B"]
}

{ "id": 103424,
  "product": {"name": "widget",
              "description": [{ "language": "English",
                                "text": "this is a widget"},
```

```
                                {"language": "Welsh",
                                 "text":  "Mae hon yn widget"}
                               ]
            },
  "price": 5.99,
  "stock": 73
}

{ "array1" : [1,2,3,4,5],
  "array2" : ["a", "b", "c"],
  "array3" : [{"a":1}, 2, "c"]
}
```

Handling of JSON Objects in Python

Both Python 2 and Python 3 have standard library support for JSON Objects.

Python treats JSON as a *serialization format* for its objects, and provided functionality to go back and forth between a JSON string and a Python object.

The mappings between the JSON syntactic constructs and the Python object types is presented below.

Decoding JSON objects into Python. When decoding JSON objects into Python, the following decoding scheme is followed.

JSON	Python
`JSON object`	dictionary
`JSON array`	list
`JSON string`	str
`JSON number`	int or float
`true`	`True`
`false`	`False`
`null`	`None`

Encoding Python objects as JSON serializations. When serializing Python objects in JSON format, the following encoding scheme is followed.

Python	JSON
dictionary	`JSON object`
list	`JSON array`
tuple	`JSON array`
int	`number`
float	`number`
`True`	`true`
`False`	`false`
`None`	`null`

json library. Python 2 and Python 3 have a standard `json` library for serialization-deserialization of JSON objects. The core functions from the library are:

function	explanation
`json.dump(obj, file, attrs)`	Serialize Python `obj` as a JSON string to a file
`json.dumps(obj, attrs)`	Serialize Python `obj` as a JSON string
`json.load(file, attrs)`	Load a JSON object/array from `file` into a Python object (return value)
`json.dumps(s, attrs)`	Transform string `s` containing JSON into a Python object (return value)

Examples. Here are some example uses.

Reading JSON data.

```
>>> import json
>>> s = '{"a":1, "b":"first", "c":[1,2,3]}'
>>> dict = json.loads(s)
>>> dict
{'c': [1, 2, 3], 'b': 'first', 'a': 1}
>>> file = open('json', "r")
>>> str = file.read()
>>> str
'{\n "id": 75,\n "name": {"first": "Mary",\n          "las": "Young"\n         },\n
"hometown": {"town": "Santa Cruz",\n              "state": "CA"\n             },
\n "magicDigits": [1,2,4,5,"nothing"]\n}\n'
>>> file.close()
>>> file=open('json',"r")
>>> d1 = json.load(file)
>>> d1
{'name': {'first': 'Mary', 'las': 'Young'}, 'id': 75, 'magicDigits': [1, 2, 4, 5, 'nothing'],
'hometown': {'state': 'CA', 'town': 'Santa Cruz'}}
>>> d1['first']
>>> d1['name']
{'first': 'Mary', 'las': 'Young'}
>>> d1['name']['first']
'Mary'
```

Creating JSON objects.

```
>>> kv = {"a":3, "b":17, "g": [1,2,3]}
>>> jsonKV = json.dumps(kv)
>>> jsonKV
'{"b": 17, "a": 3, "g": [1, 2, 3]}'
>>> fileToWrite = open('myJson', "w")
>>> json.dump(kv, fileToWrite)
>>> fileToWrite.close()
>>> fl = open('myJson', "r")
>>> text = fl.read()
>>> text
'{"b": 17, "a": 3, "g": [1, 2, 3]}'
```