JSON

JSON is a standard way of organizing text in a way that is both human-readable, and computer readable.  JSON is a standard, understood by many systems. It's very widely used in the web for client-server communication, in general for two computers to exchange data, and many other contexts.

It's plain text, so it's simple to store, it's human-readable, and can be read as string data, which you, and every programmer, and every programming language, knows how to deal with.

JSON, pronounced 'jason', stands for JavaScript object notation. It was originally designed to use with JavaScript, but it's also understood by many other programming languages, and today, is used widely by web applications, and in many other contexts.

Here's some JSON. It's plain text data. Notice it looks just like Python dictionary and list syntax! In JSON, the { } containing lists of key:value pairs are called 'objects', and comma-separated things in the [ ] are called arrays.  This example is adapted from the response to a request from OMBD (omdbapi.com)

**{**   
**"Title":"Frozen",**  
**"Year":"2013",**  
**"Rated":"PG",**  
**"Released":"27 Nov 2013",**  
**"Runtime":"102 min",**  
**"Genre":"Animation, Adventure, Comedy",**  
**"Director":"Chris Buck, Jennifer Lee",**  
**"Writer":"Jennifer Lee (screenplay), Hans Christian Andersen (story inspired by \"The Snow Queen\" by), Chris Buck (story by), Jennifer Lee (story by), Shane Morris (story by)",**  
**"Actors":[**   
**"Kristen Bell",**  
**"Tommy Lioutas",**  
**"Idina Menzel",**  
**"Jonathan Groff"**  
**],**  
**"Plot":"When the newly crowned Queen Elsa accidentally uses her power to turn things into ice to curse her home in infinite winter, her sister, Anna, teams up with a mountain man, his playful reindeer, and a snowman to change the weather condition.",**  
**"Language":"English, Icelandic",**  
**"Country":"USA",**  
**"Awards":{**   
**"Oscars":2,**  
**"Other Awards":72,**  
**"Nominations":57**  
**},**  
**"Poster":"https://images-na.ssl-images-amazon.com/images/M/MV5BMTQ1MjQwMTE5OF5BMl5BanBnXkFtZTgwNjk3MTcyMDE@.\_V1\_SX300.jpg",**  
**"Metascore":"74",**  
**"imdbRating":"7.6",**  
**"imdbVotes":"432,749",**  
**"imdbID":"tt2294629",**  
**"Type":"movie",**  
**"Response":"True"**  
**}**

As you can see, a human can read this and have a good idea what this data represents. And, the standard format means that it could be read by a computer program, and conversely, computers can easily generate JSON.

Valid JSON must follow particular syntax rules. A JSON object is surrounded with {} brackets, and is made of key:value pairs. Keys must be strings, defined with double quotes. The values can be one of the following: string, number, boolean, an array, another object. Example object for a new ITEC student:

**{    
 "school":"MCTC",   
 "Program":ITEC",   
 "Credits\_earned":11,  
 "Graduated":false  
}**

Which could also be written like this, because whitespace is ignored.

**{ "School":"MCTC", "Program":"ITEC", "Credits\_earned":11, "Graduated":false }**

A JSON array is defined like this, with square brackets, and data separated by commas. Arrays can contain strings, numbers, booleans, other arrays, or objects. For example:

**[ "ITEC 1150", "ITEC 1310", "ITEC 1425" ]**

JSON arrays are ordered, the data will always stay in the same place. You could mix data types in an array if desired, e.g. [ **2, 56.78,  "MCTC",  true,  "hello" ]** but that's unusual, it's more typical to have data all of the same type, that represents a list of a particular thing.

An object can have an array as a key's value, like so,

**{    
 "School":"MCTC",   
 "Program":"ITEC",   
 "Credits\_earned":11,  
 "Graduated":false,  
 "Classes\_taken": [ "ITEC 1150", "ITEC 1310", "ITEC 1425"]  
}**

And an array can contain an object as a value, for example, in this array of objects,

**[   
   {   
      "Class":"ITEC 1150",  
      "name":"Programming Logic"  
   },  
   {   
      "Class":"ITEC 1425",  
      "Name":"Data Communications"  
   }  
]**

And you can have objects nested in objects, nested in objects, nested in arrays, nested in objects.... as you need . So that means you can nest objects, and arrays, as you need. Here's another example, this is an error message from a website, formatted as JSON. Note the nested objects - look for the curly braces,

**{**   
**"response":{**   
**"version":"0.1",**  
**"termsofService":"http://www.wunderground.com/weather/api/d/terms.html",**  
**"features":{**   
  
**},**  
**"error":{**   
**"type":"keynotfound",**  
**"description":"this key does not exist"**  
**}**  
**}**  
**}**

If you need to write JSON, or format a lump of JSON data into a more readable format, a JSON formatter site is useful, like this one: <https://jsonformatter.curiousconcept.com/>

**JSON in Python**

JSON looks a lot like Python dictionaries and lists, so you may not be surprised to hear that Python supports turning JSON strings into the equivalent Python structures, ready for use in your program.   It also supports turning Python objects into JSON.

* **Deserializing** is turning JSON data into objects/data in your program - for example, JSON received from a HTTP request, read from a file...
* **Serializing** is turning objects in your program into JSON, either to transmit over the network, save to a file...

Python's JSON processing library docs are here; <https://docs.python.org/3/library/json.html>. The core functions you'll use most are

* json.dump() for writing Python objects to file, network, other streams
* json.dumps() (dump string) for serializing, turning Python objects to JSON strings
* json.load() for deserializing, turning JSON data, from a file or network, into Python objects
* json.loads() for deserializing, turning a JSON string, into Python objects

Some example code:

import json  
  
  
# Turn Python object into JSON strings  
phone\_data = { 'Brand' : 'Samsung', 'Model' : 'Galaxy Note 7', 'Explodes' : True }  
phone\_str = json.dumps(phone\_data) # Turn object into a JSON string  
print(phone\_str)  
  
# Write Python object to a file, as a JSON string  
json.dump(phone\_data, open('phone.json', 'w')) # Write object to a file, as a JSON string  
  
# Read JSON from a file, as Python object  
phone\_json = json.load(open('phone.json', 'r')) # phone\_json is a Python dictionary  
print(phone\_json['Brand']) # 'Samsung'  
  
# Load Python object from a JSON string  
json\_string = '''{  
 "School":"MCTC",  
 "Program":"ITEC",  
 "Credits earned":11,  
 "Graduated":false,  
 "Classes\_taken": [ "ITEC 1150", "ITEC 1310", "ITEC 1425"]  
}'''  
  
student\_data = json.loads(json\_string) # Turn String into Python objects  
  
print(student\_data['School']) # 'MCTC'  
list\_of\_classes\_taken = student\_data['Classes taken']  
for c in list\_of\_classes\_taken:  
 print('The student has taken: ' + c) # Prints each item from the Classes taken array  
  
  
# Once you have a Python object, either from a JSON file or a JSON string, can work with it in code  
# The Python object is just a regular dictorary/list, with regular Python data types, matching the structure of the JSON  
# It will be structured/nested in the same way, so in this example, a dictionary, and the value of the Classes taken key is a Python list.   
# Let's add another class that our student has taken:  
student\_data['Classes\_taken'].append('ITEC 1100')  
student\_data["Credits\_earned"] = 13  
  
# Turn objects into JSON string  
student\_json\_string = json.dumps(student\_data)  
print(student\_json\_string)  
  
# Write to a file or to some other stream (e.g. a network connection)  
f = open('student\_data.json', 'w')  
json.dump(student\_data, f)

**See also: Object serialization with JSON:** Common in other languages that don't map so neatly as JavaScript or Python do.  Here, if you know what form a JSON object is, can define a class that matches the structure of the JSON, with the same variables, and then turn your JSON into objects in your code. e.g. GSON for Java/Android, <https://github.com/google/gson>  Newtonsoft for .NET languages <http://www.newtonsoft.com/json>.

**See also: XML** - similar goals to JSON, used for many of the same purposes. XML can be more verbose, JSON's syntax and markup is simpler; but for more complex data; XML files are more concise. XML supports more features, better for large amounts of data or more complex . Used in Maven; describe GUIs (iOS, Android, Java, .NET) and **YAML**- similar purposes, less verbose than JSON, commonly seen in server configuration files.

<http://www.utilities-online.info/xmltojson/>  XML-JSON converter