

# JSON QUICK GUIDE

[http://www.tutorialspoint.com/json/json\\_quick\\_guide.htm](http://www.tutorialspoint.com/json/json_quick_guide.htm)

Copyright © tutorialspoint.com

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. Conventions used by JSON are known to programmers which include C, C++, Java, Python, Perl etc.

- JSON stands for JavaScript Object Notation.
- This format was specified by Douglas Crockford.
- This was designed for human-readable data interchange
- JSON has been extended from the JavaScript scripting language.
- JSON filename extension is **.json**
- JSON Internet Media type is **application/json**
- The Uniform Type Identifier is public.json

## Uses of JSON

- JSON is used when writing JavaScript based application which includes browser extension and websites.
- JSON format is used for serializing & transmitting structured data over network connection.
- JSON is primarily used to transmit data between server and web application.
- Web Services and APIs use JSON format to provide public data.
- JSON can be used with modern programming languages.

## Characteristics of JSON

- It is easy to read and write JSON.
- JSON is lightweight text based interchange format
- JSON is language independent.

## Simple Example in JSON

Example shows Books information stored using JSON considering language of books and their editions:

```
{
  "book": [
    {
      "id": "01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },
    {
      "id": "07",
      "language": "C++",
      "edition": "second",
      "author": "E. Balagurusamy"
    }
  ]
}
```

After understanding the above program we will try another example, let's save the below code as **json.htm**:

```
<html>
```

```

<head>
<title>JSON example</title>
<script language="javascript" >

    var object1 = { "language" : "Java", "author" : "herbert schildt" };
    document.write("<h1>JSON with JavaScript example</h1>");
    document.write("<br>");
    document.write("<h3>Language = " + object1.language+"</h3>");
    document.write("<h3>Author = " + object1.author+"</h3>");

    var object2 = { "language" : "C++", "author" : "E-Balagurusamy" };
    document.write("<br>");
    document.write("<h3>Language = " + object2.language+"</h3>");
    document.write("<h3>Author = " + object2.author+"</h3>");

    document.write("<hr />");
    document.write(object2.language + " programming language can be studied " +
    "from book written by " + object2.author);
    document.write("<hr />");

</script>
</head>
<body>
</body>
</html>

```

Now let's try to open [json.htm](#) using IE or any other javascript enabled browser, this produces the following result:

# JSON with JavaScript example

**Language = Java**

**Author = herbert schildt**

**Language = C++**

**Author = E-Balagurusamy**

---

C++ programming language can be studied from book written by E-Balagurusamy

---

## JSON - SYNTAX

Let's have a quick look on JSON basic syntax. JSON syntax is basically considered as subset of JavaScript syntax, it includes the following :

- Data is represented in name/value pairs
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by ,(comma).
- Square brackets hold arrays and values are separated by ,(comma).

Below is a simple example:

```

{
  "book": [
    {
      "id": "01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },
    {
      "id": "07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }
  ]
}

```

JSON supports following two data structures:

- **Collection of name/value pairs:** This Data Structure is supported by different programming language.
- **Ordered list of values:** It includes array, list, vector or sequence etc.

## JSON - DATATYPES

---

There are following datatypes supported by JSON format:

Type	Description
Number	double- precision floating -point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
null	empty

### Number

- It is a double precision floating -point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number.

The following table shows number types:

Type	Description
Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like .3, .9
Exponent	Exponent like e, e+, e-, E, E+, E-

---

## Syntax:

```
var json-object-name = { string : number_value, .....}
```

## Example:

Example showing Number Datatype, value should not be quoted:

```
var obj = {marks: 97}
```

## String

- It is a sequence of zero or more double quoted Unicode characters with backslash escaping.
- Character is a single character string i.e. a string with length 1.

The table shows string types:

Type	Description
"	double quotation
\	reverse solidus
/	solidus
b	backspace
f	form feed
n	new line
r	carriage return
t	horizontal tab
u	four hexadecimal digits

## Syntax:

```
var json-object-name = { string : "string value", .....}
```

## Example:

Example showing String Datatype:

```
var obj = {name: 'Amit'}
```

## Boolean

It includes true or false values.

## Syntax:

```
var json-object-name = { string : true/false, .....}
```

## Example:

```
var obj = {name: 'Amit', marks: 97, distinction: true}
```

## Array

- It is an ordered collection of values.
- 
- These are enclosed square brackets which means that array begins with `[` and ends with `]`.
- 
- The values are separated by `,` (comma).
- 
- Array indexing can be started at 0 or 1.
- 
- Arrays should be used when the key names are sequential integers.
- 

## Syntax:

```
[ value, .....]
```

## Example:

Example showing array containing multiple objects:

```
{
  "books": [
    { "language": "Java" , "edition": "second" },
    { "language": "C++" , "lastName": "fifth" },
    { "language": "C" , "lastName": "third" }
  ]
}
```

## Object

- It is an unordered set of name/value pairs.
- 
- Object are enclosed in curly braces that is it starts with `{` and ends with `}`.
- 
- Each name is followed by `:` (colon) and the name/value pairs are separated by `,` (comma).
- 
- The keys must be strings and should be different from each other.
- 
- Objects should be used when the key names are arbitrary strings
-

## Syntax:

```
{ string : value, ..... }
```

## Example:

Example showing Object:

```
{  
  "id": "011A",  
  "language": "JAVA",  
  "price": 500,  
}
```

## Whitespace

It can be inserted between any pair of tokens. It can be added to make code more readable. Example shows declaration with and without whitespace:

## Syntax:

```
{string:"  ",.....}
```

## Example:

```
var i= "  sachin";  
var j = "  saurav"
```

## null

It means empty type.

## Syntax:

```
null
```

## Example:

```
var i = null;  
  
if(i==1)  
{  
  document.write("<h1>value is 1</h1>");  
}  
else  
{  
  document.write("<h1>value is null</h1>");  
}
```

## JSON Value

It includes:

- number (integer or floating point)
- 
- string
-

- boolean
- 
- array
- 
- object
- 
- null
- 

## Syntax:

```
String | Number | Object | Array | TRUE | FALSE | NULL
```

## Example:

```
var i =1;
var j = "sachin";
var k = null;
```

# JSON - OBJECTS

## Creating Simple Objects

JSON objects can be created with Javascript. Let's us see various ways of creating JSON objects using Javascript:

- Creation of an empty Object:

```
var JSONObj = {};
```

- Creation of new Object:

```
var JSONObj = new Object();
```

- Creation of an object with attribute **bookname** with value in string, attribute **price** with numeric value. Attributes are accessed by using '.' Operator:

```
var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
```

This is an example which shows creation of an object in javascript using JSON, save the below code as **json\_object.htm**:

```
<html>
<head>
<title>Creating Object JSON with JavaScript</title>
<script language="javascript" >

    var JSONObj = { "name" : "tutorialspoint.com", "year" : 2005 };
    document.write("<h1>JSON with JavaScript example</h1>");
    document.write("<br>");
    document.write("<h3>Website Name="+JSONObj.name+"</h3>");
    document.write("<h3>Year="+JSONObj.year+"</h3>");

</script>
</head>
<body>
```

```
</body>
</html>
```

Now let's try to open [json\\_object.htm](#) using IE or any other javascript enabled browser, this produces the following result:

# JSON with JavaScript example

**Website Name=tutorialspoint.com**

**Year=2005**

## Creating Array Objects

Below example shows creation of an array object in javascript using JSON, save the below code as **json\_array\_object.htm**:

```
<html>
<head>
<title>Creation of array object in javascript using JSON</title>
<script language="javascript" >

document.writeln("<h2>JSON array object</h2>");

var books = { "Pascal" : [
    { "Name" : "Pascal Made Simple", "price" : 700 },
    { "Name" : "Guide to Pascal", "price" : 400 }
],
  "Scala" : [
    { "Name" : "Scala for the Impatient", "price" : 1000 },
    { "Name" : "Scala in Depth", "price" : 1300 }
  ]
}

var i = 0
document.writeln("<table border='2'><tr>");
for(i=0;i<books.Pascal.length;i++)
{
    document.writeln("<td>");
    document.writeln("<table border='1' width=100 >");
    document.writeln("<tr><td><b>Name</b></td><td width=50>"
    + books.Pascal[i].Name+"</td></tr>");
    document.writeln("<tr><td><b>Price</b></td><td width=50>"
    + books.Pascal[i].price + "</td></tr>");
    document.writeln("</table>");
    document.writeln("</td>");
}

for(i=0;i<books.Scala.length;i++)
{
    document.writeln("<td>");
    document.writeln("<table border='1' width=100 >");
    document.writeln("<tr><td><b>Name</b></td><td width=50>"
    + books.Scala[i].Name+"</td></tr>");
    document.writeln("<tr><td><b>Price</b></td><td width=50>"
    + books.Scala[i].price+"</td></tr>");
    document.writeln("</table>");
    document.writeln("</td>");
}
document.writeln("</tr></table>");
</script>
</head>
```



```
<body>
</body>
</html>
```

Now let's try to open [json\\_array\\_object.htm](#) using IE or any other javascript enabled browser, this produces the following result:

## JSON array object

<b>Name</b>	Pascal Made Simple	<b>Name</b>	Guide to Pascal	<b>Name</b>	Scala for the Impatient	<b>Name</b>	Scala in Depth
<b>Price</b>	700	<b>Price</b>	400	<b>Price</b>	1000	<b>Price</b>	1300

## JSON - SCHEMA

---

JSON Schema is a specification for JSON based format for defining structure of JSON data. It was written under IETF draft which expired in 2011. JSON Schema:

- Describes your existing data format.
- Clear, human- and machine-readable documentation.
- Complete structural validation, useful for automated testing.
- Complete structural validation, validating client-submitted data.

### JSON Schema Validation Libraries

There are several validators currently available for different programming languages. Currently the most complete and compliant JSON Schema validator available is JSV

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

### JSON Schema Example

Following is a basic JSON schema which covers a classical product catalog description:

---

```

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },
    "name": {
      "description": "Name of the product",
      "type": "string"
    },
    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
  },
  "required": ["id", "name", "price"]
}

```

Let's check various important keywords which can be used in this schema:

Keywords	Description
\$schema	The \$schema keyword states that this schema is written according to the draft v4 specification.
title	You will use this to give a title to your schema
description	A little description of the schema
type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.
minimum	This is the constraint to be put on the value and represents minimum acceptable value.
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
maximum	This is the constraint to be put on the value and represents maximum acceptable value.
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	The length of a string instance is defined as the maximum number of its characters.
minLength	The length of a string instance is defined as the minimum number of its characters.
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

You can check a <http://json-schema.org> for complete list of keywords which can be used in defining JSON schema. Above schema can be used to test the validity of the below given JSON code:

```
[
  {
    "id": 2,
    "name": "An ice sculpture",
    "price": 12.50,
  },
  {
    "id": 3,
    "name": "A blue mouse",
    "price": 25.50,
  }
]
```

## JSON - COMPARISON WITH XML

---

JSON and XML are human readable formats and are language independent. They both have support for creation, reading and decoding in real world situations. We can compare JSON with XML based on the following factors:

### Verbose

XML is more verbose than JSON, so it's faster to write JSON for humans.

### Arrays Usage

XML is used to describe structured data which doesn't include arrays whereas JSON include arrays.

### Parsing

JavaScript's *eval* method parses JSON. When applied to JSON, eval returns the described object.

### Example

This shows individual examples of XML and JSON:

### JSON

```
{
  "company": Volkswagen,
  "name": "Vento",
  "price": 800000
}
```

### XML

```
<car>
  <company>Volkswagen</company>
  <name>Vento</name>
  <price>800000</price>
</car>
```

## JSON WITH PHP

---

This chapter will teach you how to encode and decode JSON objects using PHP programming language. Let's start with preparing environment to start our programming with PHP for JSON.

### Environment

As of PHP 5.2.0, the JSON extension is bundled and compiled into PHP by default.

### JSON Functions

Function	Libraries
json_encode	Returns the JSON representation of a value
json_decode	Decodes a JSON string
json_last_error	Returns the last error occurred

## Encoding JSON in PHP (json\_encode)

PHP json\_encode() function is used for encoding JSON in PHP. This function returns the JSON representation of a value on success or FALSE on failure.

### Syntax:

```
string json_encode ( $value [, $options = 0 ] )
```

### Parameters:

- **value:** The value being encoded. This function only works with UTF-8 encoded data.
- **options:** This optional value is a bitmask consisting of JSON\_HEX\_QUOT, JSON\_HEX\_TAG, JSON\_HEX\_AMP, JSON\_HEX\_APOS, JSON\_NUMERIC\_CHECK, JSON\_PRETTY\_PRINT, JSON\_UNESCAPED\_SLASHES, JSON\_FORCE\_OBJECT

### Example

The following example shows how to convert an arrays into JSON with PHP:

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
echo json_encode($arr);
?>
```

While executing, this will produce following result:

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

The following example shows how PHP objects can be converted into JSON:

```
<?php
class Emp {
    public $name = "";
    public $hobbies = "";
    public $birthdate = "";
}
$e = new Emp();
$e->name = "sachin";
$e->hobbies = "sports";
$e->birthdate = date('m/d/Y h:i:s a', strtotime("8/5/1974 12:20:03"));

echo json_encode($e);
?>
```

While executing, this will produce following result:

```
{"name":"sachin","hobbies":"sports","birthdate":"08/05/1974 12:20:03 pm"}
```

## Decoding JSON in PHP (json\_decode)

PHP json\_decode() function is used for decoding JSON in PHP. This function returns the value decoded from json to appropriate PHP type.

## Syntax:

```
mixed json_decode ($json [, $assoc = false [, $depth = 512 [, $options = 0 ]]])
```

## Parameters:

- **json\_string**: It is encoded string which must be UTF-8 encoded data
- **assoc**: It is a boolean type parameter, when set to TRUE, returned objects will be converted into associative arrays.
- **depth**: It is an integer type parameter which specifies recursion depth
- **options**: It is an integer type bitmask of JSON decode, JSON\_BIGINT\_AS\_STRING is supported.

## Example

The following example shows how PHP can be used to decode JSON objects:

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

var_dump(json_decode($json));
var_dump(json_decode($json, true));
?>
```

While executing, this will produce following result:

```
object(stdClass)#1 (5) {
  ["a"] => int(1)
  ["b"] => int(2)
  ["c"] => int(3)
  ["d"] => int(4)
  ["e"] => int(5)
}

array(5) {
  ["a"] => int(1)
  ["b"] => int(2)
  ["c"] => int(3)
  ["d"] => int(4)
  ["e"] => int(5)
}
```

## JSON WITH PERL

This chapter will teach you how to encode and decode JSON objects using Perl programming language. Let's start with preparing environment to start our programming with Perl for JSON.

### Environment

Before you start with encoding and decoding JSON using Perl, you will need to install JSON module which can be obtained from CPAN. Once you downloaded JSON-2.53.tar.gz or any other latest version, follow the following steps:

```
$tar xvfz JSON-2.53.tar.gz
$cd JSON-2.53
$perl Makefile.PL
$make
$make install
```

## JSON Functions

Function	Libraries
encode_json	Converts the given Perl data structure to a UTF-8 encoded, binary string.
decode_json	Decodes a JSON string.
to_json	Converts the given Perl data structure to a json string.
from_json	Expects a json string and tries to parse it, returning the resulting reference.
convert_blessed	Use this function with true value so that Perl can use TO_JSON method on the object's class to convert an object into JSON.

## Encoding JSON in Perl (encode\_json)

Perl encode\_json() function converts the given Perl data structure to a UTF-8 encoded, binary string.

### Syntax:

```
$json_text = encode_json ($perl_scalar );

or

$json_text = JSON->new->utf8->encode($perl_scalar);
```

## Example

The following example shows arrays under JSON with Perl:

```
#!/usr/bin/perl
use JSON;

my %rec_hash = ('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
my $json = encode_json \%rec_hash;
print "$json\n";
```

While executing, this will produce following result:

```
{"e":5,"c":3,"a":1,"b":2,"d":4}
```

The following example shows how Perl objects can be converted into JSON:

```
#!/usr/bin/perl

package Emp;
sub new
{
    my $class = shift;
    my $self = {
        name => shift,
        hobbies => shift,
        birthdate => shift,
    };
    bless $self, $class;
    return $self;
}
sub TO_JSON { return { %{ shift() } }; }

package main;
use JSON;

my $JSON = JSON->new->utf8;
$JSON->convert_blessed(1);
```

```
$e = new Emp( "sachin", "sports", "8/5/1974 12:20:03 pm");
$json = $JSON->encode($e);
print "$json\n";
```

While executing, this will produce following result:

```
{"birthdate":"8/5/1974 12:20:03 pm","name":"sachin","hobbies":"sports"}
```

## Decoding JSON in Perl (decode\_json)

Perl `decode_json()` function is used for decoding JSON in Perl. This function returns the value decoded from json to appropriate Perl type.

### Syntax:

```
$perl_scalar = decode_json $json_text

or

$perl_scalar = JSON->new->utf8->decode($json_text)
```

## Example

The following example shows how Perl can be used to decode JSON objects. Here you will need to install `Data::Dumper` module if you already do not have it on your machine.

```
#!/usr/bin/perl
use JSON;
use Data::Dumper;

$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

$text = decode_json($json);
print Dumper($text);
```

While executing, this will produce following result:

```
$VAR1 = {
    'e' => 5,
    'c' => 3,
    'a' => 1,
    'b' => 2,
    'd' => 4
};
```

## JSON WITH PYTHON

This chapter will teach you how to encode and decode JSON objects using Python programming language. Let's start with preparing environment to start our programming with Python for JSON.

### Environment

Before you start with encoding and decoding JSON using Python, you will need to install any of the JSON modules available. For this tutorial I downloaded and installed [Demjson](#) as follows:

```
$tar xvfz demjson-1.6.tar.gz
$cd demjson-1.6
$python setup.py install
```

## JSON Functions

Function	Libraries
----------	-----------

encode	Encodes the Python object into a JSON string representation.
decode	Decodes a JSON-encoded string into a Python object

## Encoding JSON in Python (encode)

Python `encode()` function encodes the Python object into a JSON string representation.

### Syntax:

```
demjson.encode(self, obj, nest_level=0)
```

### Example

The following example shows arrays under JSON with Python

```
#!/usr/bin/python
import demjson

data = [ { 'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e' : 5 } ]

json = demjson.encode(data)
print json
```

While executing, this will produce following result:

```
[{"a":1,"b":2,"c":3,"d":4,"e":5}]
```

## Decoding JSON in Python (decode)

Python can use `demjson.decode()` function for decoding JSON. This function returns the value decoded from json to appropriate Python type.

### Syntax:

```
demjson.decode(self, txt)
```

### Example

The following example shows how Python can be used to decode JSON objects.

```
#!/usr/bin/python
import demjson

json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

text = demjson.decode(json)
print text
```

While executing, this will produce following result:

```
{u'a': 1, u'c': 3, u'b': 2, u'e': 5, u'd': 4}
```

## JSON WITH RUBY

This chapter will teach you how to encode and decode JSON objects using Ruby programming language. Let's start with preparing environment to start our programming with Ruby for JSON.

### Environment



Before you start with encoding and decoding JSON using Ruby, you will need to install any of the JSON modules available for Ruby. You may need to install Ruby gem, but if you are running latest version of Ruby then you must have gem already installed on your machine, otherwise let's follow the following single step assuming you already have gem installed:

```
$gem install json
```

## Parsing JSON using Ruby

The following example shows that the first 2 keys hold string values and the last 3 keys hold arrays of strings. Let's keep the following content in a file called **input.json**

```
{
  "President": "Alan Isaac",
  "CEO": "David Richardson",

  "India": [
    "Sachin Tendulkar",
    "Virender Sehwag",
    "Gautam Gambhir",
  ],

  "Srilanka": [
    "Lasith Malinga",
    "Angelo Mathews",
    "Kumar Sangakkara"
  ],

  "England": [
    "Alastair Cook",
    "Jonathan Trott",
    "Kevin Pietersen"
  ]
}
```

The following is ruby program which will be used to parse above mentioned JSON document:

```
#!/usr/bin/ruby
require 'rubygems'
require 'json'
require 'pp'

json = File.read('input.json')
obj = JSON.parse(json)

pp obj
```

While executing, this will produce following result:

```
{"President"=>"Alan Isaac",
 "CEO"=>"David Richardson",

 "India"=>
  ["Sachin Tendulkar", "Virender Sehwag", "Gautam Gambhir"],

 "Srilanka"=>
  ["Lasith Malinga ", "Angelo Mathews", "Kumar Sangakkara"],

 "England"=>
  ["Alastair Cook", "Jonathan Trott", "Kevin Pietersen"]
}
```

## JSON WITH JAVA

---

This chapter will teach you how to encode and decode JSON objects using Java programming language. Let's start with preparing environment to start our programming with Java for JSON.

## Environment

Before you start with encoding and decoding JSON using Java, you will need to install any of the JSON modules available. For this tutorial I downloaded and installed [JSON.simple](#) and add the location of **json-simple-1.1.1.jar** file to environment variable CLASSPATH:

## Mapping between JSON and Java entities

JSON.simple maps entities from the left side to the right side while decoding or parsing, and maps entities from the right to the left while encoding.

JSON	Java
string	java.lang.String
number	java.lang.Number
true false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

While decoding, default concrete class of *java.util.List* is *org.json.simple.JSONArray* and default concrete class of *java.util.Map* is *org.json.simple.JSONObject*.

## Encoding JSON in Java

Following is a simple example to encode a JSON object using Java JSONObject which is a subclass of java.util.HashMap. No ordering is provided. If you need strict ordering of elements use JSONValue.toJSONString( map ) method with ordered map implementation such as java.util.LinkedHashMap.

```
import org.json.simple.JSONObject;

class JsonEncodeDemo
{
    public static void main(String[] args)
    {
        JSONObject obj = new JSONObject();

        obj.put("name", "foo");
        obj.put("num", new Integer(100));
        obj.put("balance", new Double(1000.21));
        obj.put("is_vip", new Boolean(true));

        System.out.print(obj);
    }
}
```

While compile and executing above program, this will produce following result:

```
{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}
```

Following is another example which shows JSON object streaming using Java JSONObject:

```
import org.json.simple.JSONObject;

class JsonEncodeDemo
{
    public static void main(String[] args)
    {
```

```

JSONObject obj = new JSONObject();

obj.put("name", "foo");
obj.put("num", new Integer(100));
obj.put("balance", new Double(1000.21));
obj.put("is_vip", new Boolean(true));

StringWriter out = new StringWriter();
obj.writeJSONString(out);

String jsonText = out.toString();
System.out.print(jsonText);
}
}

```

While compile and executing above program, this will produce following result:

```

{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}

```

## Decoding JSON in Java

Following example makes use of **JSONObject** and **JSONArray** where JSONObject is a java.util.Map and JSONArray is a java.util.List, so you can access them with standard operations of Map or List.

```

import org.json.simple.JSONObject;
import org.json.simple.JSONArray;
import org.json.simple.parser.ParseException;
import org.json.simple.parser.JSONParser;

class JsonDecodeDemo
{
    public static void main(String[] args)
    {
        JSONParser parser=new JSONParser();
        String s = "[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}]";
        try{
            Object obj = parser.parse(s);
            JSONArray array = (JSONArray)obj;
            System.out.println("The 2nd element of array");
            System.out.println(array.get(1));
            System.out.println();

            JSONObject obj2 = (JSONObject)array.get(1);
            System.out.println("Field \"1\"");
            System.out.println(obj2.get("1"));

            s = "{}";
            obj = parser.parse(s);
            System.out.println(obj);

            s= "[5,]";
            obj = parser.parse(s);
            System.out.println(obj);

            s= "[5,,2]";
            obj = parser.parse(s);
            System.out.println(obj);
        }catch(ParseException pe){
            System.out.println("position: " + pe.getPosition());
            System.out.println(pe);
        }
    }
}

```

While compile and executing above program, this will produce following result:

```

The 2nd element of array
{"1":{"2":{"3":{"4":[5,{"6":7}]}}}

```

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

## JSON WITH AJAX

---

Ajax is Asynchronous JavaScript and XML which is used on client side as group of interrelated web development techniques in order to create asynchronous web applications. According to Ajax model, web applications can send data and retrieve data from a server asynchronously without interfering with the display, behavior of existing page.

Many developers use JSON to pass AJAX updates between client and server. Websites updating live sports scores can be considered as an example of AJAX. If these scores have to be updated on the website, then they must be stored on the server so that the webpage can retrieve the score when it is required. This is where we can make use of JSON formatted data.

Any data that is updated using AJAX can be stored using the JSON format on web server. Ajax is used so that javascript can retrieve these JSON files when necessary, they parse them and then does of the two:

- Store the parsed values in variables for further processing before displaying them on the webpage
- It directly assign the data to the DOM elements in the webpage, so that it gets displayed on the website.

### Example

The below code shows JSON with Ajax, save it in **ajax.htm** file. Here loading function loadJSON() will be used asynchronously to upload JSON data.

```
<html>
<head>
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
<script type="application/javascript">
function loadJSON()
{
    var data_file = "http://www.tutorialspoint.com/json/data.json";
    var http_request = new XMLHttpRequest();
    try{
        // Opera 8.0+, Firefox, Chrome, Safari
        http_request = new XMLHttpRequest();
    }catch (e){
        // Internet Explorer Browsers
        try{
            http_request = new ActiveXObject("Msxml2.XMLHTTP");
        }catch (e) {
            try{
                http_request = new ActiveXObject("Microsoft.XMLHTTP");
            }catch (e){
                // Something went wrong
                alert("Your browser broke!");
                return false;
            }
        }
    }
    http_request.onreadystatechange = function(){
        if (http_request.readyState == 4 )
        {
            // Javascript function JSON.parse to parse JSON data
            var jsonObj = JSON.parse(http_request.responseText);

            // jsonObj variable now contains the data structure and can
            // be accessed as jsonObj.name and jsonObj.country.
            document.getElementById("Name").innerHTML = jsonObj.name;
            document.getElementById("Country").innerHTML = jsonObj.country;
        }
    }
    http_request.open("GET", data_file, true);
    http_request.send();
}
```

```

}
</script>
<title>tutorialspoint.com JSON</title>
</head>
<body>
<h1>Cricketer Details</h1>
<table >
<tr><th>Name</th><th>Country</th></tr>
<tr><td><div >Sachin</div></td>
<td><div >India</div></td></tr>
</table>

<div >
<button type="button" onclick="loadJSON()">Update Details </button>
</div>
</body>
</html>

```

Following is the input file **data.json** having data in JSON format which will be uploaded asynchronously when we click **Update Detail** button. This file is being kept in <http://www.tutorialspoint.com/json/>

```

{"name": "brett", "country": "Australia"}

```

Above HTML code will generate following screen, where you can check AJAX in action:

## CRICKETER DETAILS

---

Name	Country
------	---------

Update Details

When you click on **Update Detail** button, you should get a result something as follows, you can try it yourself [JSON with AJAX](#) provided your browser supports Javascript.

## CRICKETER DETAILS

---

Name	Country
------	---------