



# JSON

javascript object notation

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json. The JSON filename extension is .json.

This tutorial will help you understand JSON and its use within various programming languages such as PHP, PERL, Python, Ruby, Java, etc.

## Audience

---

This tutorial has been designed to help beginners understand the basic functionality of JavaScript Object Notation (JSON) to develop the data interchange format. After completing this tutorial, you will have a good understanding of JSON and how to use it with JavaScript, Ajax, Perl, etc.

## Prerequisites

---

Before proceeding with this tutorial, you should have a basic understanding of the web application's work over HTTP and we assume that you have a basic knowledge of JavaScript.

## Copyright & Disclaimer

---

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

# Table of Contents

---

About the Tutorial.....	i
Audience .....	i
Prerequisites .....	i
Copyright & Disclaimer.....	i
Table of Contents .....	ii
1. JSON – OVERVIEW .....	1
Uses of JSON .....	1
Characteristics of JSON.....	1
Simple Example in JSON .....	1
2. JSON – SYNTAX.....	4
3. JSON – DATATYPES.....	5
Number.....	5
String .....	6
Boolean.....	7
Array .....	7
Object .....	8
Whitespace .....	8
null.....	9
JSON Value.....	9
4. JSON – OBJECTS.....	10
Creating Simple Objects .....	10
Creating Array Objects .....	11

5.	JSON – SCHEMA .....	13
	JSON Schema Validation Libraries .....	13
	JSON Schema Example .....	14
6.	JSON – COMPARISON WITH XML.....	17
7.	JSON – JSON WITH PHP .....	18
	Environment .....	18
	JSON Functions.....	18
	Encoding JSON in PHP (json_encode) .....	18
	Decoding JSON in PHP (json_decode).....	19
8.	JSON – JSON WITH PERL.....	21
	Environment .....	21
	JSON Functions.....	21
	Encoding JSON in Perl (encode_json) .....	22
	Decoding JSON in Perl (decode_json) .....	23
9.	JSON – JSON WITH PYTHON .....	25
	Environment .....	25
	JSON Functions.....	25
	Encoding JSON in Python (encode).....	25
	Decoding JSON in Python (decode) .....	26
10.	JSON – JSON WITH RUBY .....	27
	Environment .....	27
	Parsing JSON using Ruby .....	27

11. JSON – JSON WITH JAVA.....	29
Environment .....	29
Mapping between JSON and Java entities.....	29
Encoding JSON in Java .....	30
Decoding JSON in Java.....	31
12. JSON – JSON WITH AJAX.....	33

# 1. JSON – Overview

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

## Uses of JSON

---

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

## Characteristics of JSON

---

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

## Simple Example in JSON

---

The following example shows how to use JSON to store information related to books based on their topic and edition.

```
{  
  "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 that 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

You can refer to JSON Objects chapter for more information on JSON objects.



## 2. JSON – Syntax

Let's have a quick look at the basic syntax of JSON. JSON syntax is basically considered as a 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 the following two data structures:

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

### 3. JSON – DataTypes

JSON format supports the following data types:

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 the 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 various special characters that you can use in strings of a JSON document:

Type	Description
"	double quotation
\	backslash
/	forward slash
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 in 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.
- Objects are enclosed in curly braces that is, it starts with '{' and ends with '}'.
- Each name is followed by ':'(colon) and the key/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 a code more readable. Example shows declaration with and without whitespace:

### Syntax

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

### Example

```
var obj1 = {"name": "Sachin Tendulkar"}  
var obj2 = {"name": "SauravGanguly"}
```

## 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;
```

# 4. JSON – Objects

## Creating Simple Objects

JSON objects can be created with JavaScript. Let us see the various ways of creating JSON objects using JavaScript:

- Creation of an empty Object:

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

- Creation of a new Object:

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

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

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

This is an example that 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. It produces the following result:

## JSON with JavaScript example

Website Name=tutorialspoint.com

Year=2005

### Creating Array Objects

The following 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. It 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

## 5. JSON – Schema

JSON Schema is a specification for JSON based format for defining the 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

Given below 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 the various important keywords that 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 the complete list of keywords that can be used in defining a JSON schema. The above schema can be used to test the validity of the following JSON code:

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

## 6. 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 is faster to write JSON for programmers.

### Arrays Usage

XML is used to describe the 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

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>
```

# 7. JSON – JSON with PHP

This chapter covers how to encode and decode JSON objects using PHP programming language. Let's start with preparing the 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 array 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 the following result:

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

The following example shows how the 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', "8/5/1974 12:20:03 p");
    $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 the 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 an 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, it will produce the 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)
}
```

## 8. JSON – JSON with Perl

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

### Environment

---

Before you start encoding and decoding JSON using Perl, you 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 steps mentioned below:

```
$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 the 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";

```

On executing, it will produce the 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 an 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);
```

On executing, it will produce following result:

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

# 9. JSON – JSON with Python

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

## Environment

Before you start with encoding and decoding JSON using Python, you need to install any of the JSON modules available. For this tutorial we have downloaded and installed Demjson (<http://deron.meranda.us/python/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 the 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 an 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
```

On executing, it will produce the following result:

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

# 10. JSON – JSON with Ruby

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

## Environment

---

Before you start with encoding and decoding JSON using Ruby, you 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"
  ]
}

```

Given below is a Ruby program that will be used to parse the above mentioned JSON document:

```

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

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

pp obj

```

On executing, it will produce the 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"]
}

```

# 11. JSON – JSON with Java

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

## Environment

---

Before you start with encoding and decoding JSON using Java, you need to install any of the JSON modules available. For this tutorial we have downloaded and installed JSON.simple (<https://code.google.com/p/json-simple/>) and have added the location of **json-simple-1.1.1.jar** file to the 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

On decoding, the default concrete class of *java.util.List* is *org.json.simple.JSONArray* and the 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 the 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);
    }
}
```

On compiling and executing the above program the following result will be generated:

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

Following is another example that shows a 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);
    }
}

```

On compiling and executing the above program the following result is generated:

```

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

```

## Decoding JSON in Java

The 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);
    }
}
}

```

On compiling and executing the above program, the following result will be generated:

```

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

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

```

## 12. JSON – JSON with Ajax

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

Many developers use JSON to pass AJAX updates between the client and the 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 the web server. AJAX is used so that JavaScript can retrieve these JSON files when necessary, parse them, and perform one of the following operations:

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

### Example

The following code shows JSON with AJAX. Save it as **ajax.htm** file. Here the loading function loadJSON() is 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 class="src">
<tr><th>Name</th><th>Country</th></tr>
<tr><td><div id="Name">Sachin</div></td>
<td><div id="Country">India</div></td></tr>
</table>

<div class="central">

```

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

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

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

The above HTML code will generate the following screen, where you can check AJAX in action:

### Cricketer Details

Name	Country
Sachin	India

### Update Details:

When you click on the **Update Detail** button, you should get a result something as follows. You can try JSON with AJAX yourself, provided your browser supports Javascript.

### Cricketer Details

Name	Country
Brett	Australia