JSON
http://www.json.org/

국민대학교 컴퓨터공학부

강 승 식

JSON (JavaScript Object Notation)

- A lightweight data-interchange format

- It is easy for humans to read and write.
- It is easy for machines to parse and generate.
- It is based on a subset of the JavaScript Programming Language: Standard ECMA-262 3rd Edition – Dec. 1999.

- JSON is a text format that is
  - completely language independent
  - but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

- These properties make JSON an ideal data-interchange language.

- open-standard format
- human-readable text to transmit data objects consisting of attribute-value pairs
- replacing XML

- JSON is built on two structures:
  - A collection of name-value pairs
    - object, record, struct, dictionary, hash table, keyed list, or associative array
  - An ordered list of values
    - array, vector, list, or sequence

자료구조가 연관배열 배열
두가지가 있음

- These are universal data structures.
    - Virtually all modern programming languages support them in one form or another.
    - It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

JSON Syntax

- Derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

- JSON syntax is a subset of the JavaScript syntax.

JSON Values

- A number (integer or floating point)
- A string (in double quotes)
- A Boolean (true or false)

- An array (in square brackets)
- An object (in curly braces)

- null

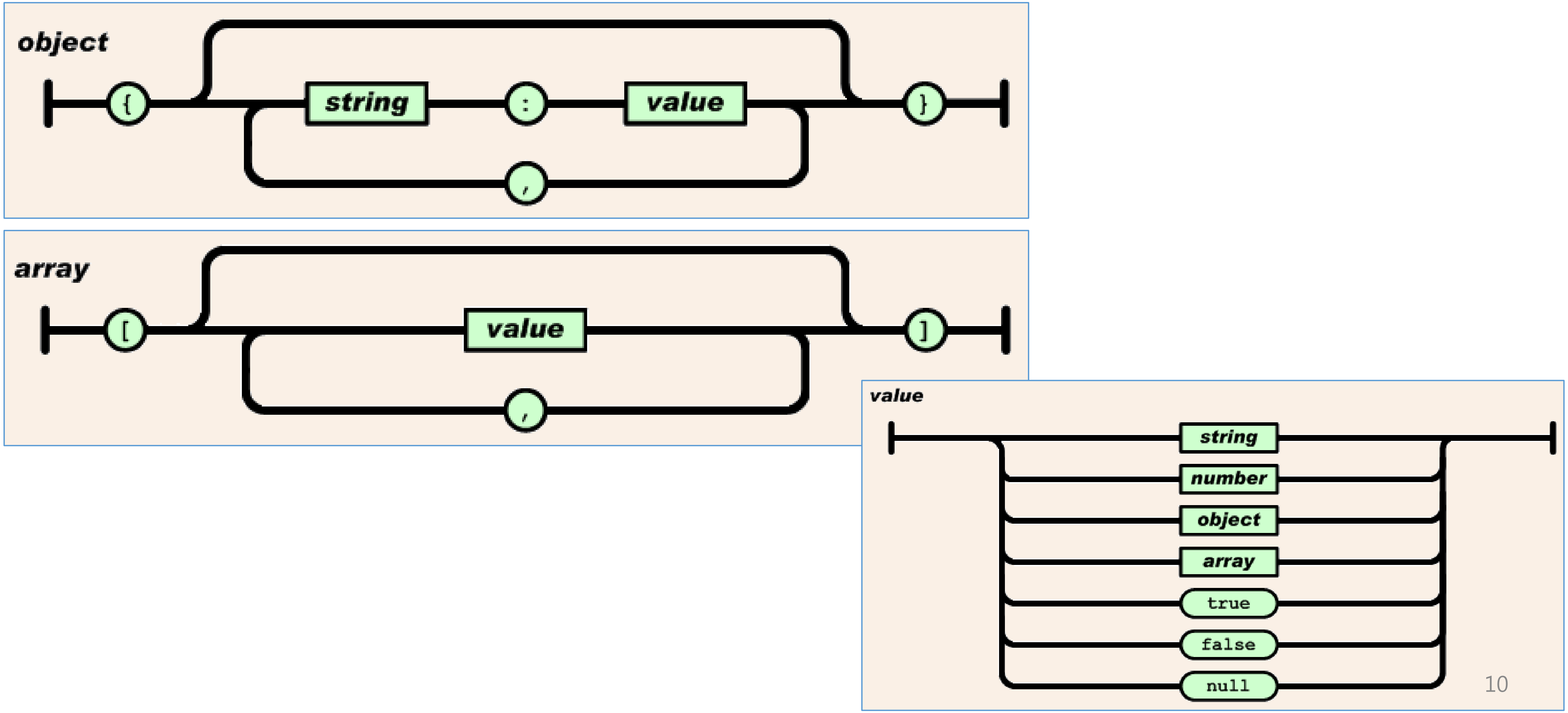
Object and array

- object
    - unordered set of name-value pairs
    - begins with '{' and ends with '}'
    - each name is followed by ':' and the name-value pairs are separated by ','

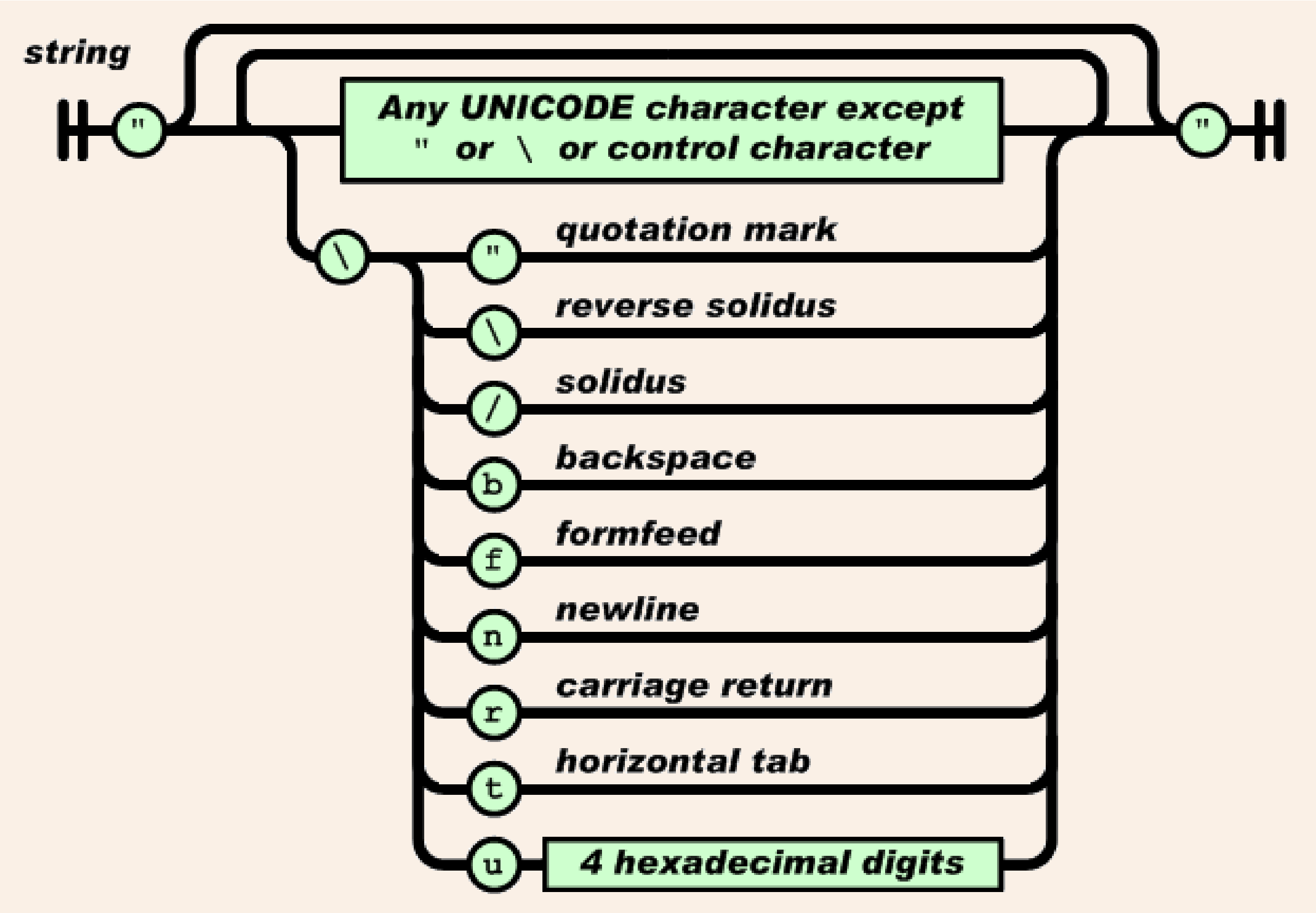
- array
    - ordered collection of values
    - begins with '[' and ends with ']'
    - values are separated by ','

```
<object>              context free grammer
            { }
            { <members> }
<members>
            <pair>
            <pair> , <members>
<pair>
            <string> : <value>

<array>      array라는 nonterminal
            [ ]
            [ <elements> ]
<elements>           elements는 value이거나
            <value>  value value value~~~
            <value> , <elements>
<value>
            <string>   " "
            <number>  정수 실수
            <object>
            <array>
            true
            false
            null
```

Syntax Diagram

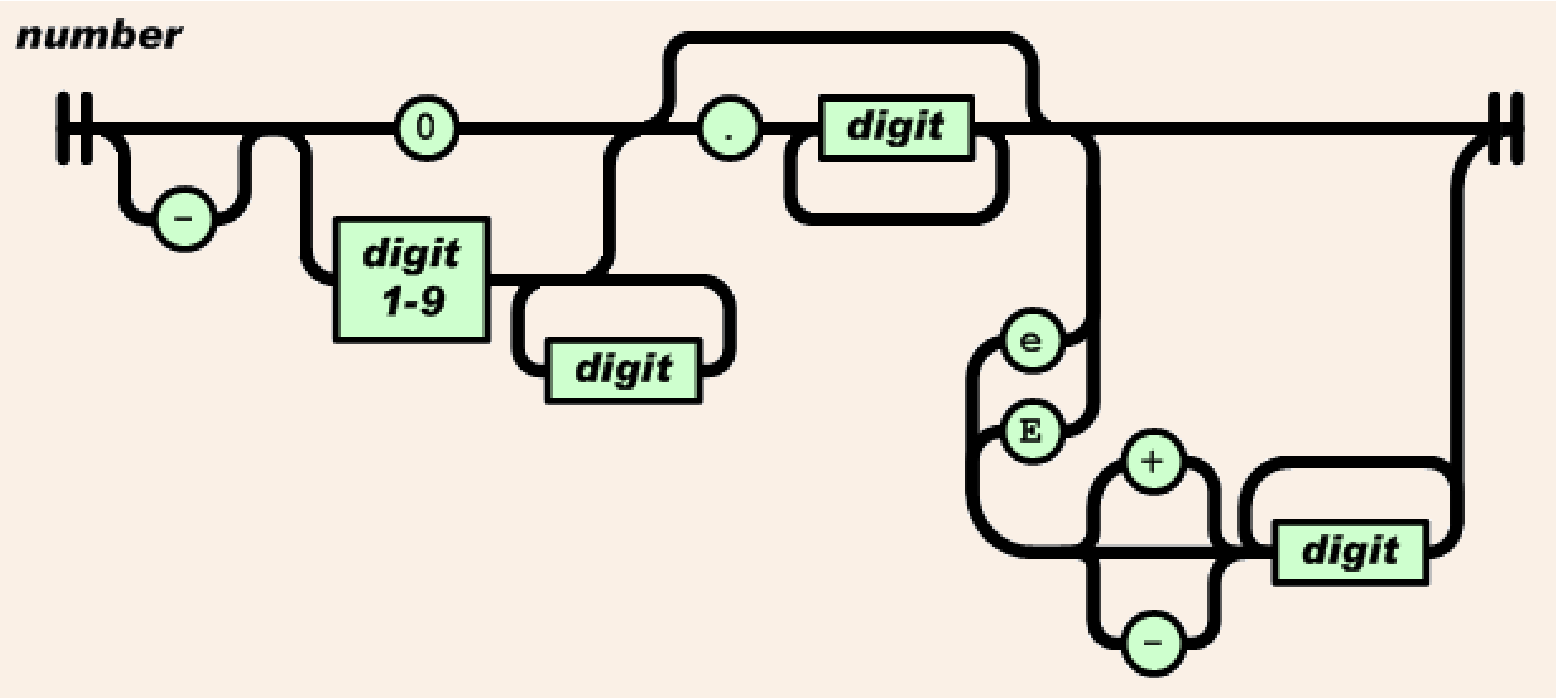


String



```
string
        ""
        " chars "
chars
        char
        char chars
char
        any-Unicode-character-
            except-"-or-₩-or-
            control-character
        ₩"
        ₩₩
        ₩/
        ₩b
        ₩f
        ₩n
        ₩r
        ₩t
        ₩u four-hex-digits
```

Number



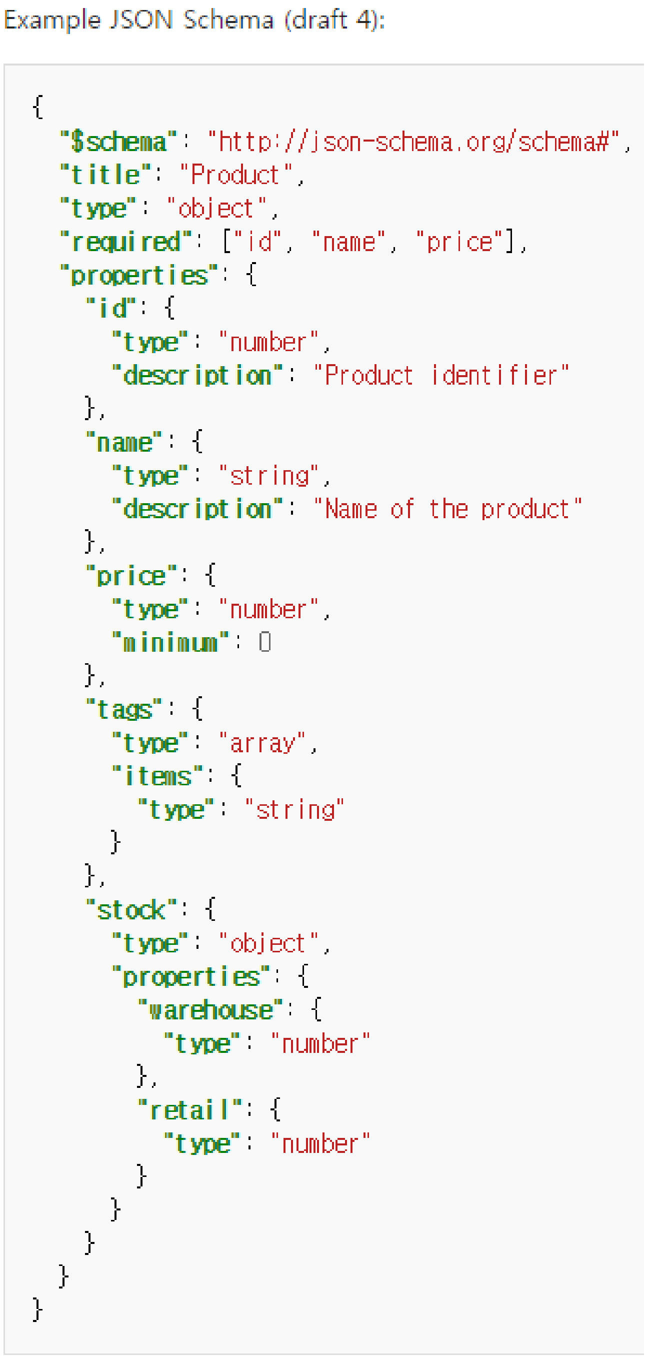
```
number
        int
        int frac
        int exp
        int frac exp
int
        digit
        digit1-9 digits
        - digit
        - digit1-9 digits
frac
        . digits
exp
        e digits
digits
        digit
        digit digits
e
        e
        e+
        e−
        E
        E+
        E−
```

JSON Files

- The file type for JSON files is ".json“

- The MIME type for JSON text is "application/json"

JSON Schema



Example JSON Schema (draft 4):

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

렉스가 생성규칙
만들고 야크가
파싱성공하면
파스트리를 만들어줘야함
파스트리는 만드는 과정도 과제에 포함
시켜야함

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

순서대로 인식시키면
{,",id,",:, 1, "," 등등

Example 1.

```
{
    "이름": "테스트",
    "나이": 25,
    "성별": "여",
    "주소": "서울특별시 양천구 목동",
    "특기": ["농구", "도술"],
    "가족관계": {"#": 2, "아버지": "홍판서", "어머니": "춘섬"},
    "회사": "경기 안양시 만안구 안양7동";
}
```

Example 2.

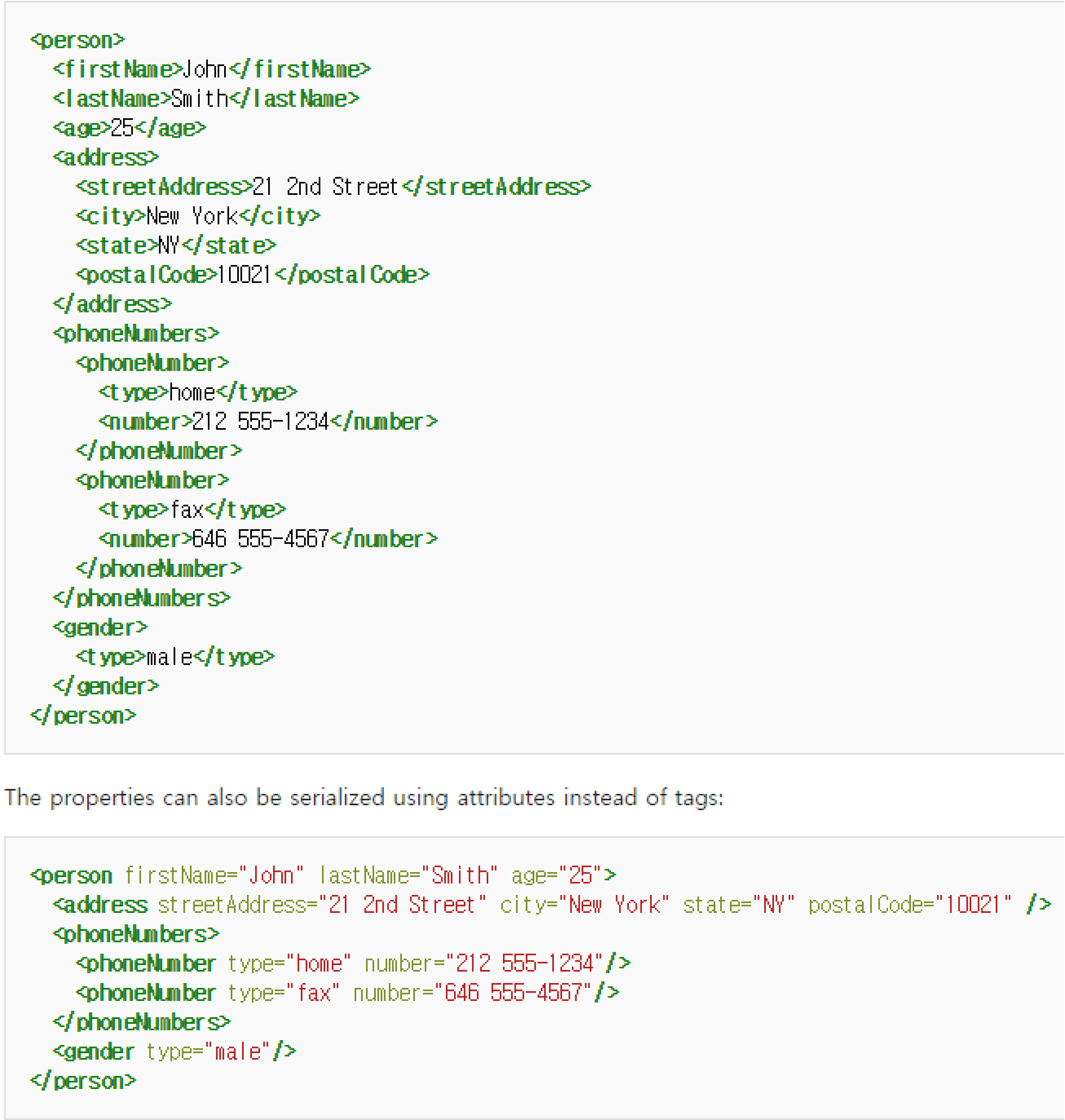
```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
```

```
 "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [ ],
  "spouse": null
}
```

YAML sample

```
---
firstName: John
lastName: Smith
age: 25
address:
  streetAddress: 21 2nd Street
  city: New York
  state: NY
  postalCode: 10021
phoneNumber:
- type: home
  number: 212 555-1234
- type: fax
  number: 646 555-4567
gender:
  type: male
```

XML samples

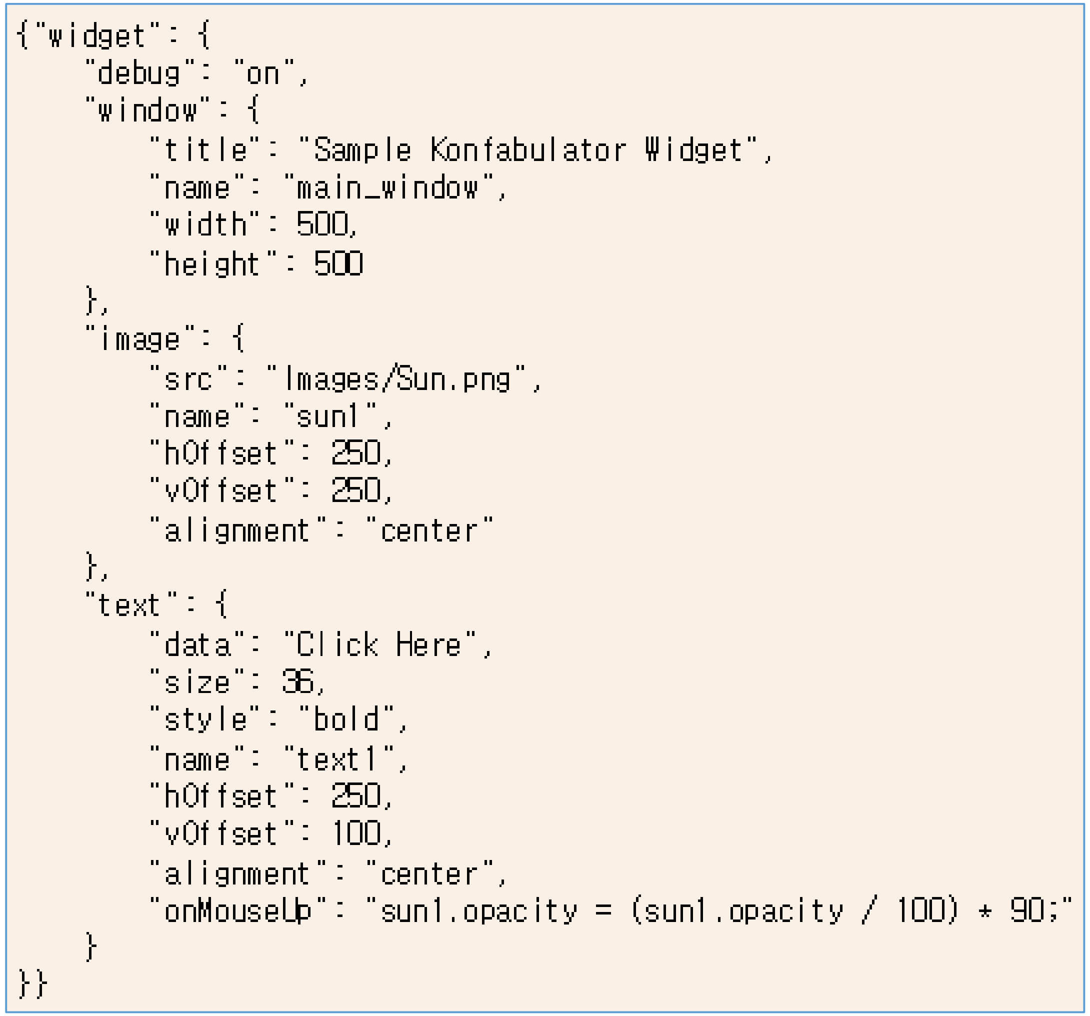


```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>
      <type>home</type>
      <number>212 555-1234</number>
    </phoneNumber>
    <phoneNumber>
      <type>fax</type>
      <number>646 555-4567</number>
    </phoneNumber>
  </phoneNumbers>
  <gender>
    <type>male</type>
  </gender>
</person>
```

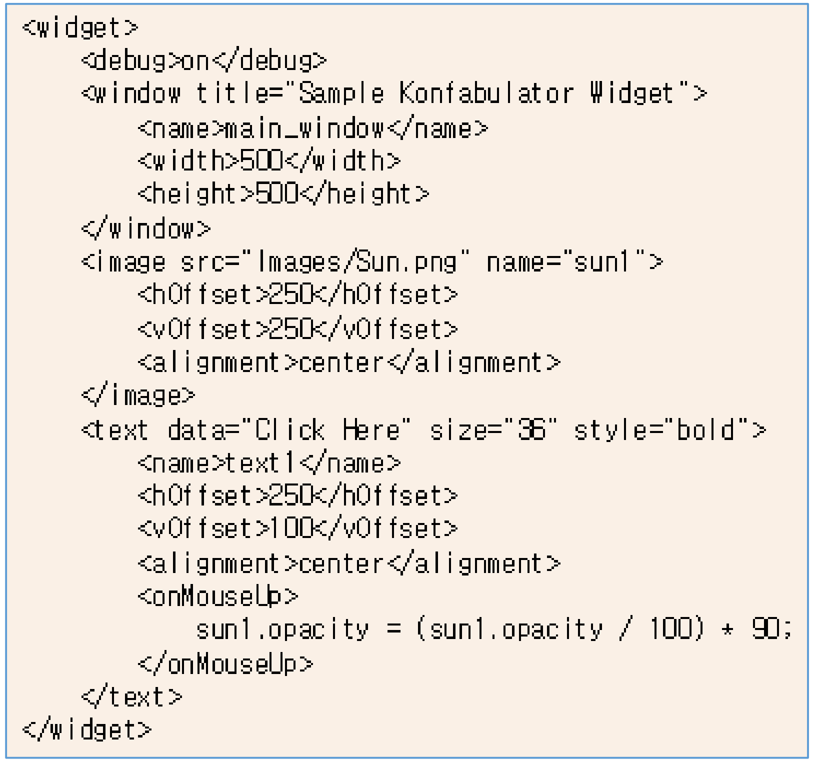
The properties can also be serialized using attributes instead of tags:

```
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" />
  <phoneNumbers>
    <phoneNumber type="home" number="212 555-1234"/>
    <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
  <gender type="male"/>
</person>
```

JSON vs. XML: http://json.org/example.html



```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
    }
}}
```



```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <vOffset>250</vOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36" style="bold">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
        <onMouseUp>
            sun1.opacity = (sun1.opacity / 100) * 90;
        </onMouseUp>
    </text>
</widget>
```

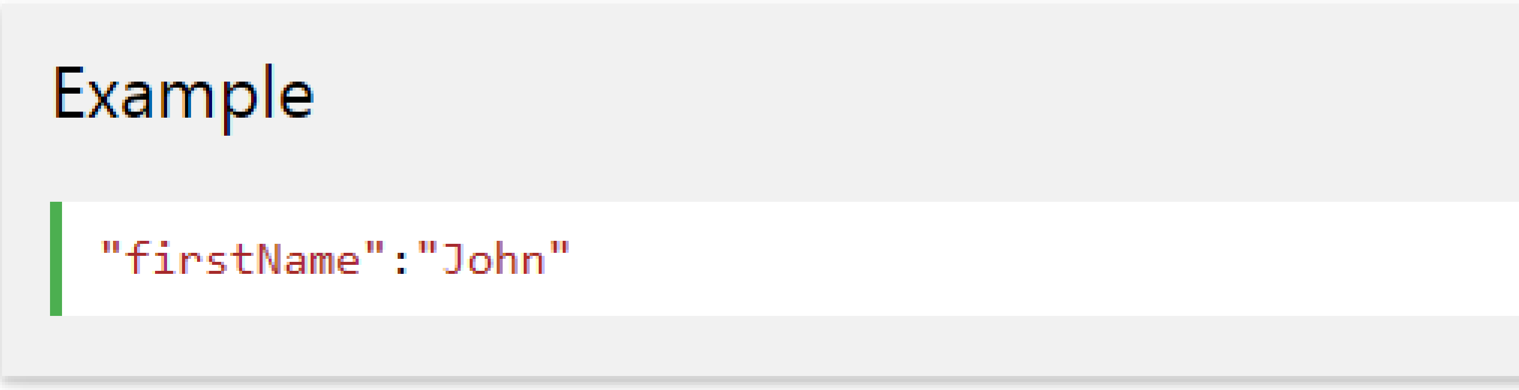
```
{"menu": {
    "id": "file",
    "value": "File",
    "popup": {
         "menuitem": [
              {"value": "New", "onclick": CreateNewDoc()"},
              {"value": "Open", "onclick": "OpenDoc()"},
              {"value": "Close", "onclick": "CloseDoc()"}
         ]
     }
}}
```

```
<menu id="file" value="File">
    <popup>
         <menuitem value="New" onclick="CreateNewDoc()" />
         <menuitem value="Open" onclick="OpenDoc()" />
         <menuitem value="Close" onclick="CloseDoc()" />
    </popup>
</menu>
```

JSON and JavaScript

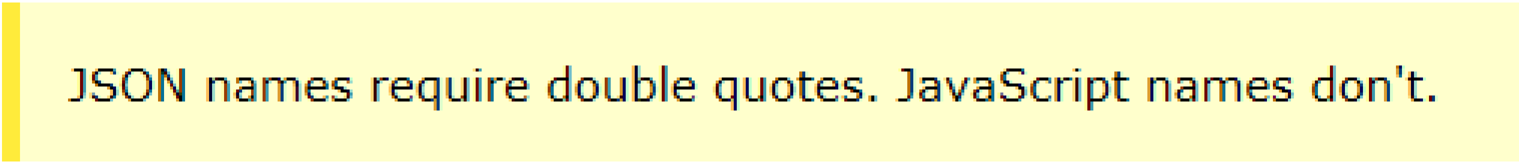
JSON Data - A Name and a Value

- field name (in double quotes), colon, value:



Example

```
"firstName":"John"
```

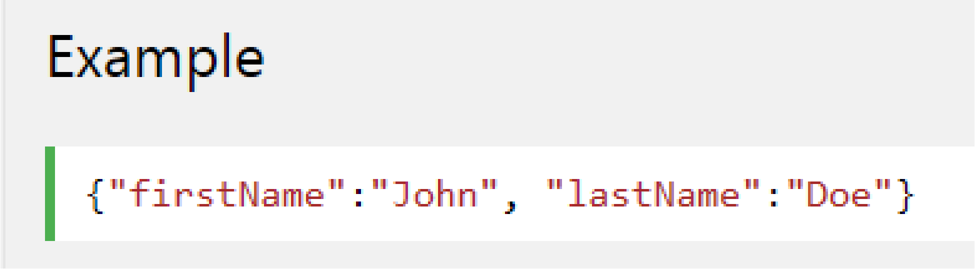


JSON names require double quotes. JavaScript names don't.

- http://www.w3schools.com/js/js_json_syntax.asp

Object and array

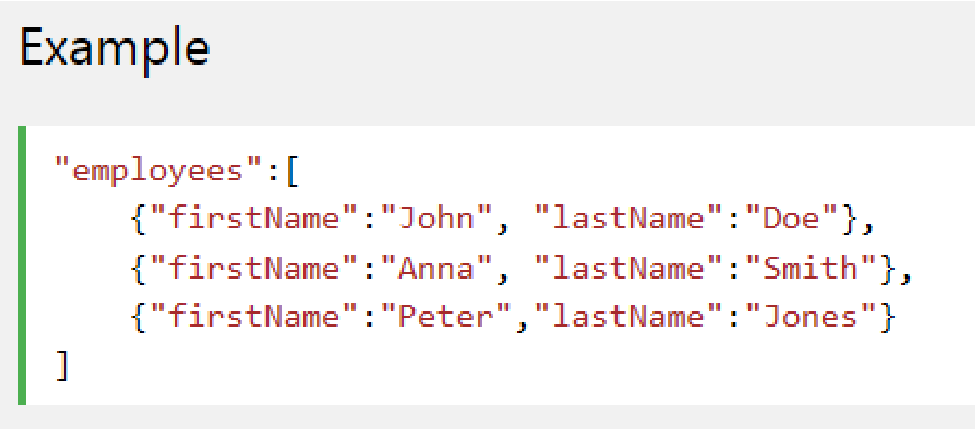
- JSON objects are written inside curly braces.



Example

```
{"firstName":"John", "lastName":"Doe"}
```

- JSON arrays are written inside square brackets.

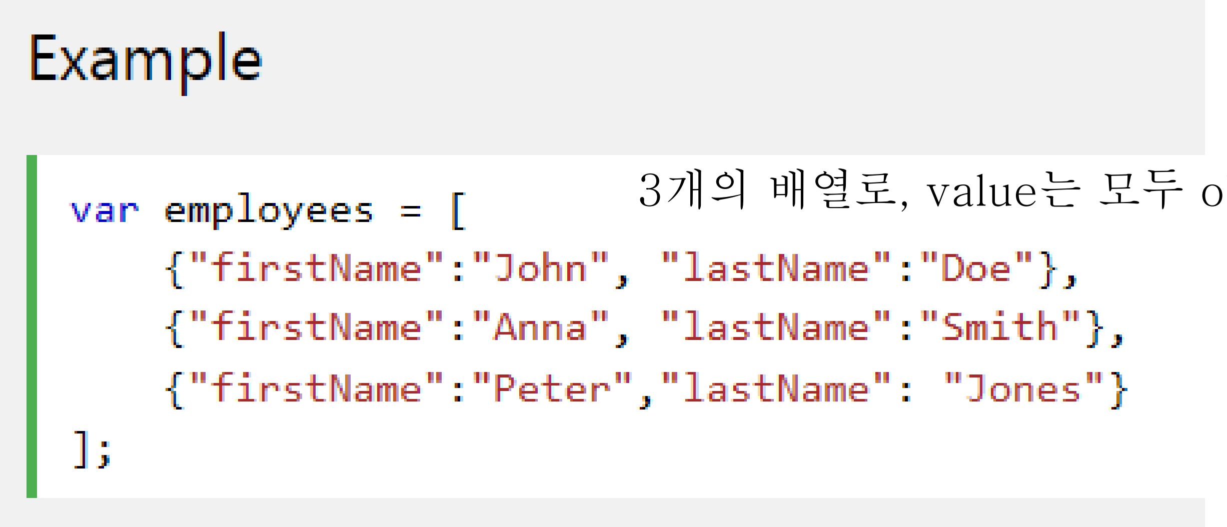


Example

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
]
```

JSON Uses JavaScript Syntax

- In JavaScript



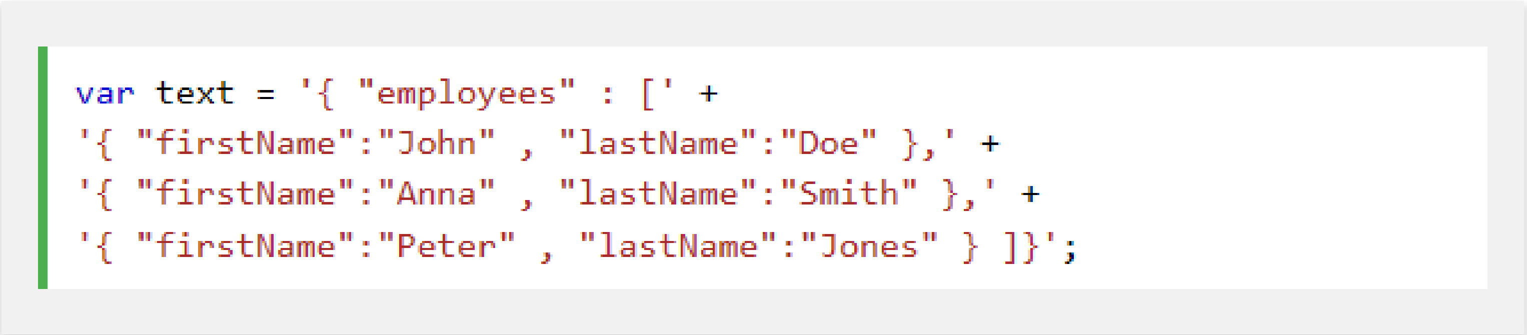
Example

```
var employees = [
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName": "Jones"}
];
```

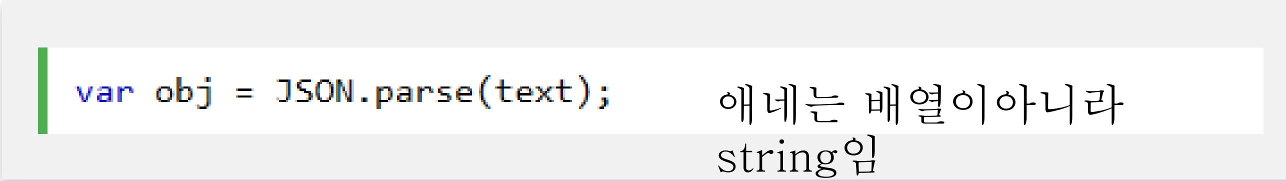
3개의 배열로, value는 모두 object로 되어있음

```
// returns John Doe
employees[0]["firstName"] + " " + employees[0]["lastName"];
employees[0].firstName = "Gilbert";
employees[0]["firstName"] = "Gilbert";
```

JSON.parse() can use the eval() function

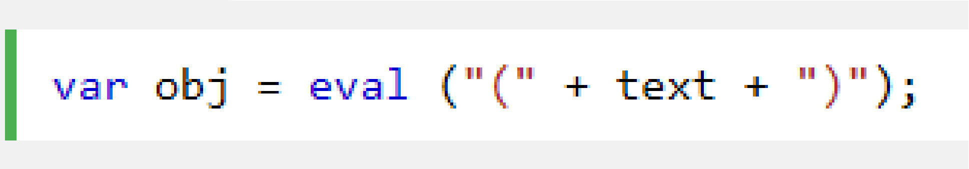


```
var text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

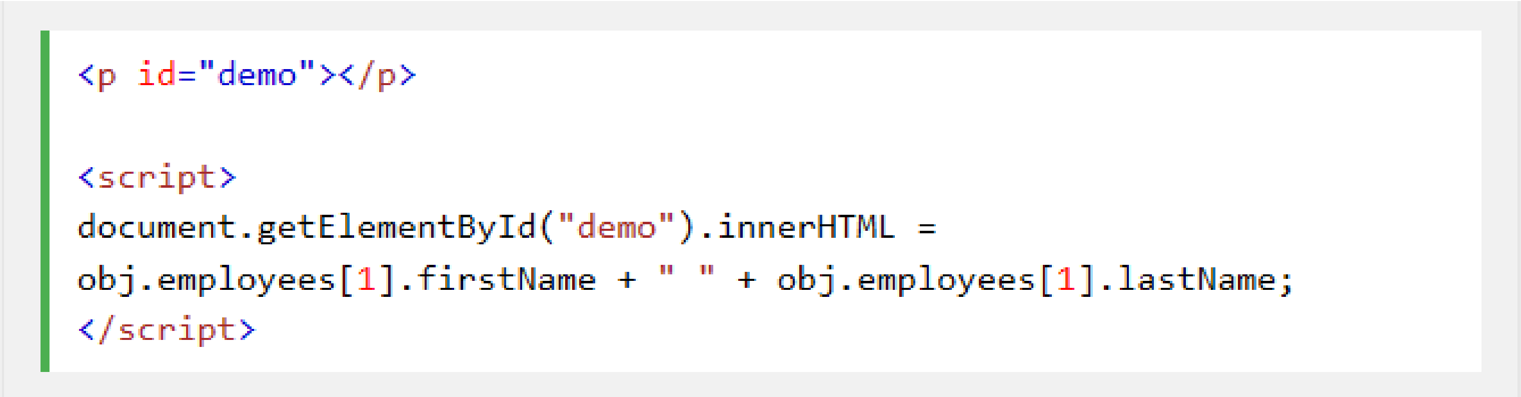


```
var obj = JSON.parse(text);
```

애네는 배열이아니라
string임



```
var obj = eval ("(" + text + ")");
```



```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

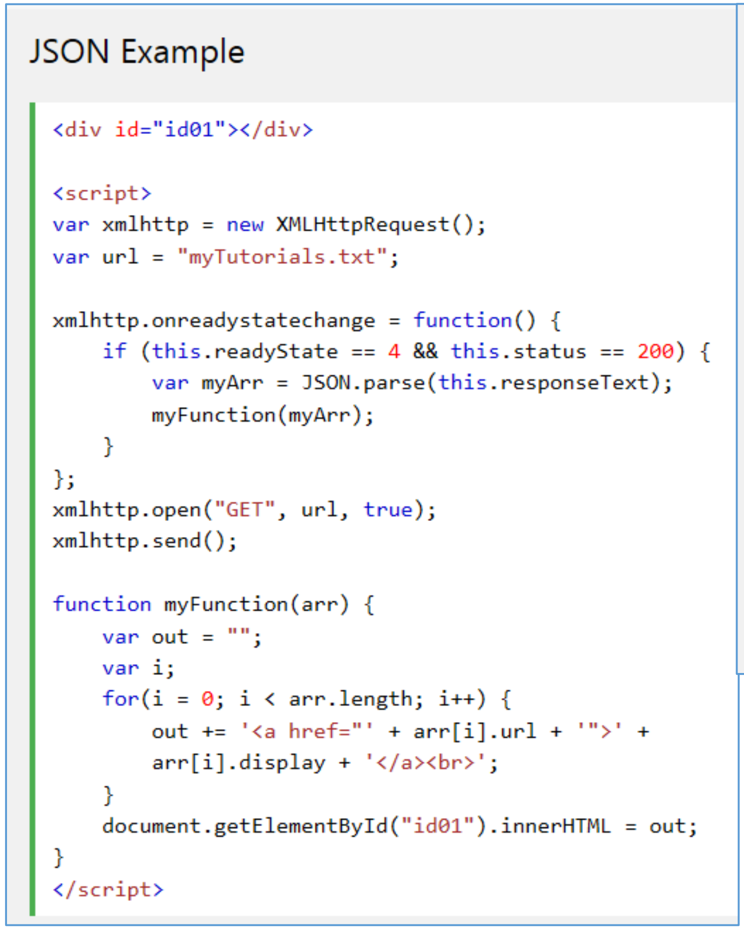
Web Browsers Support

- Firefox 3.5
- Internet Explorer 8
- Chrome
- Opera 10
- Safari 4

- For older browsers, a JavaScript library is available at https://github.com/douglascrockford/JSON-js.

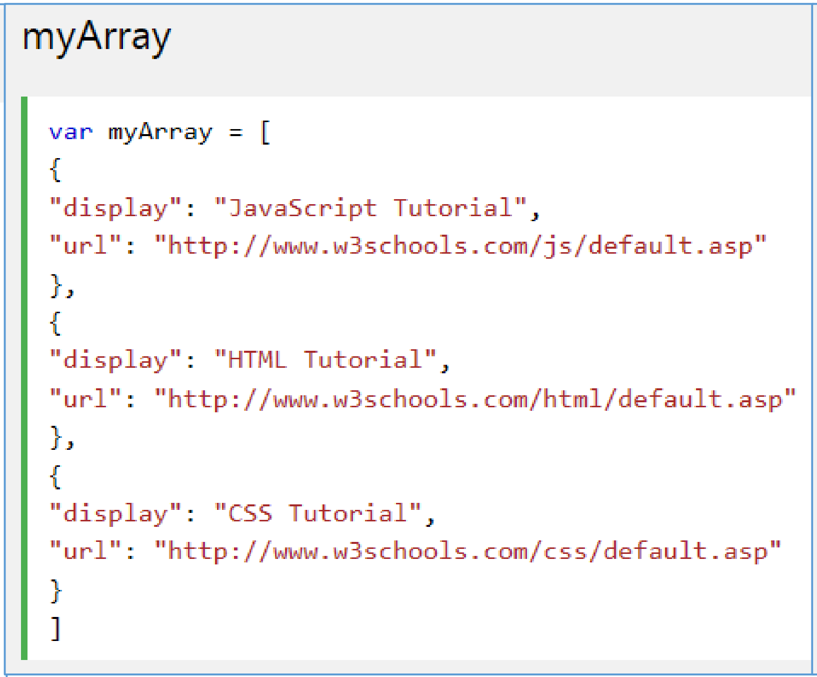
JSON Http Request

- http://www.w3schools.com/js/js_json_http.asp



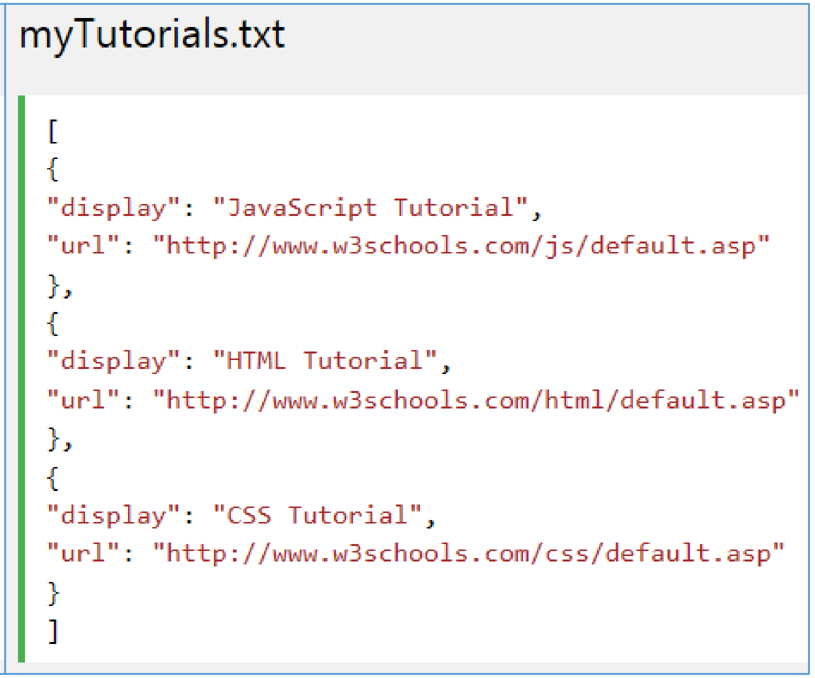
JSON Example

```
<div id="id01"></div>

<script>
var xmlhttp = new XMLHttpRequest();
var url = "myTutorials.txt";

xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        var myArr = JSON.parse(this.responseText);
        myFunction(myArr);
    }
};
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' +
        arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
</script>
```



myArray

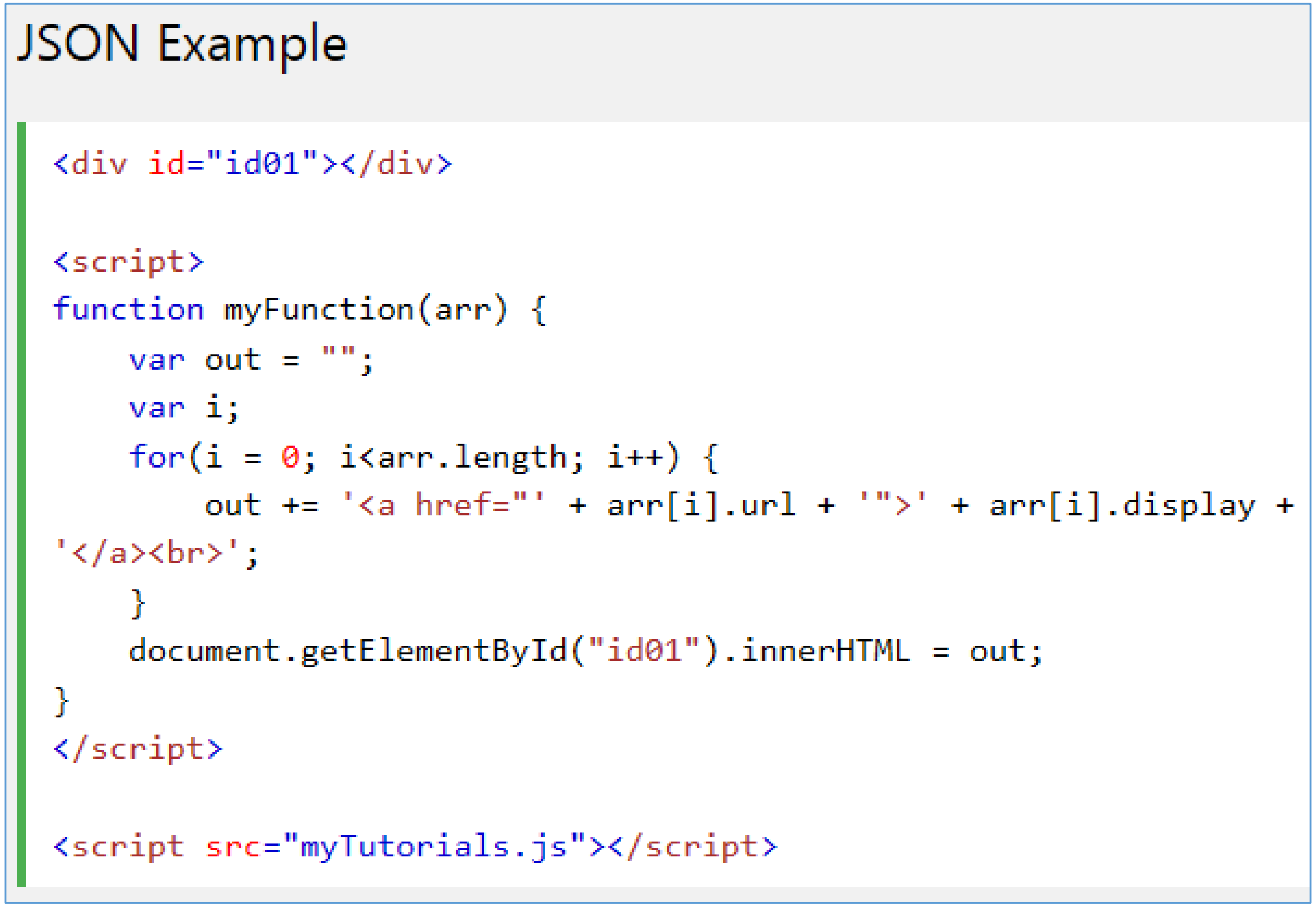
```
var myArray = [
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]
```



myTutorials.txt

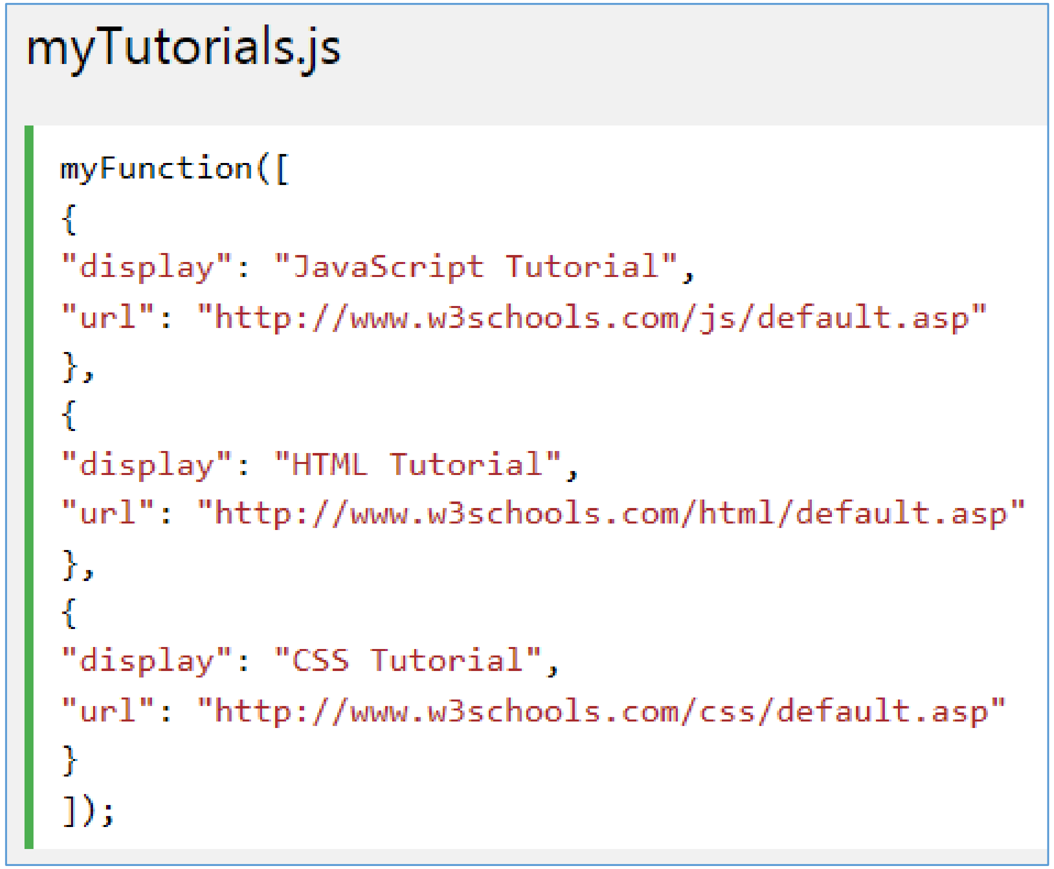
```
[
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]
```

JSON Function Files



JSON Example

```
<div id="id01"></div>

<script>
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i<arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display +
'</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
</script>

<script src="myTutorials.js"></script>
```

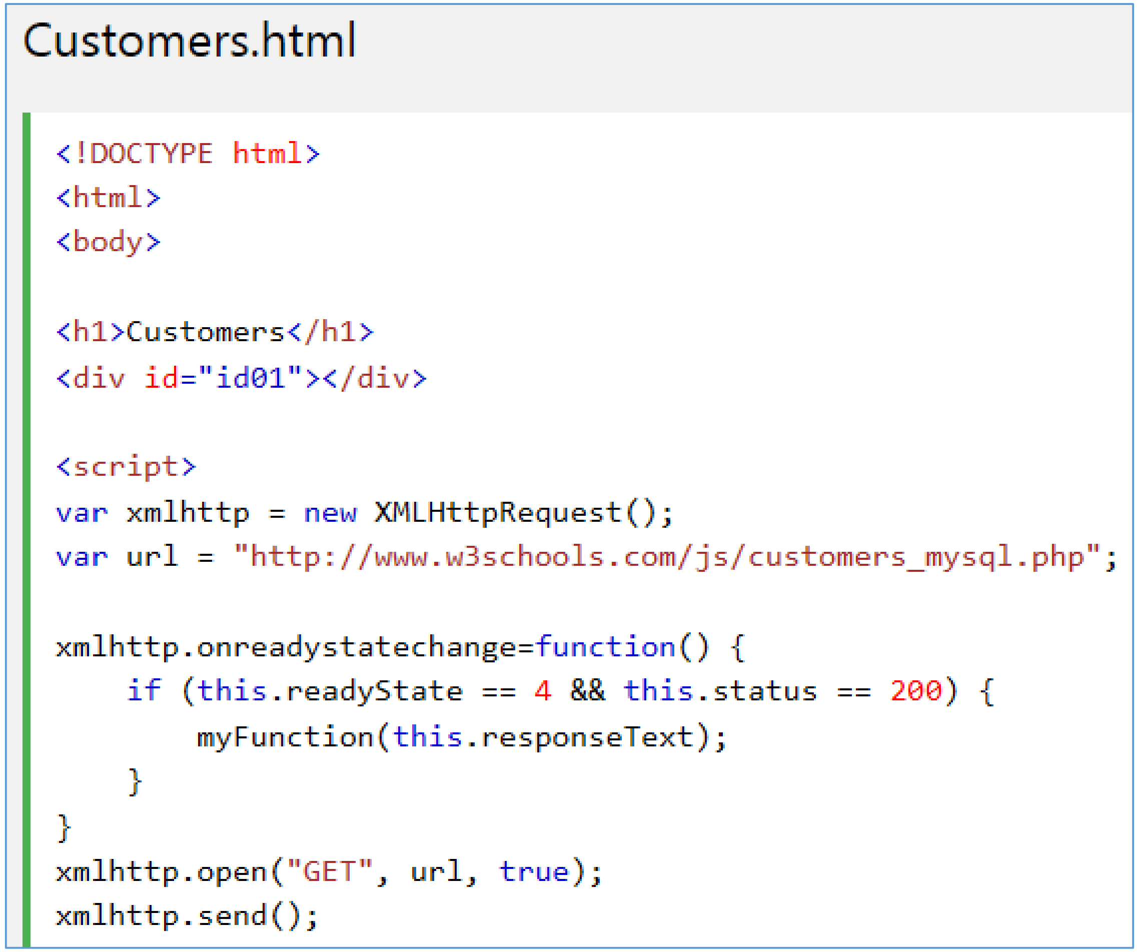


myTutorials.js

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
}
]);
```

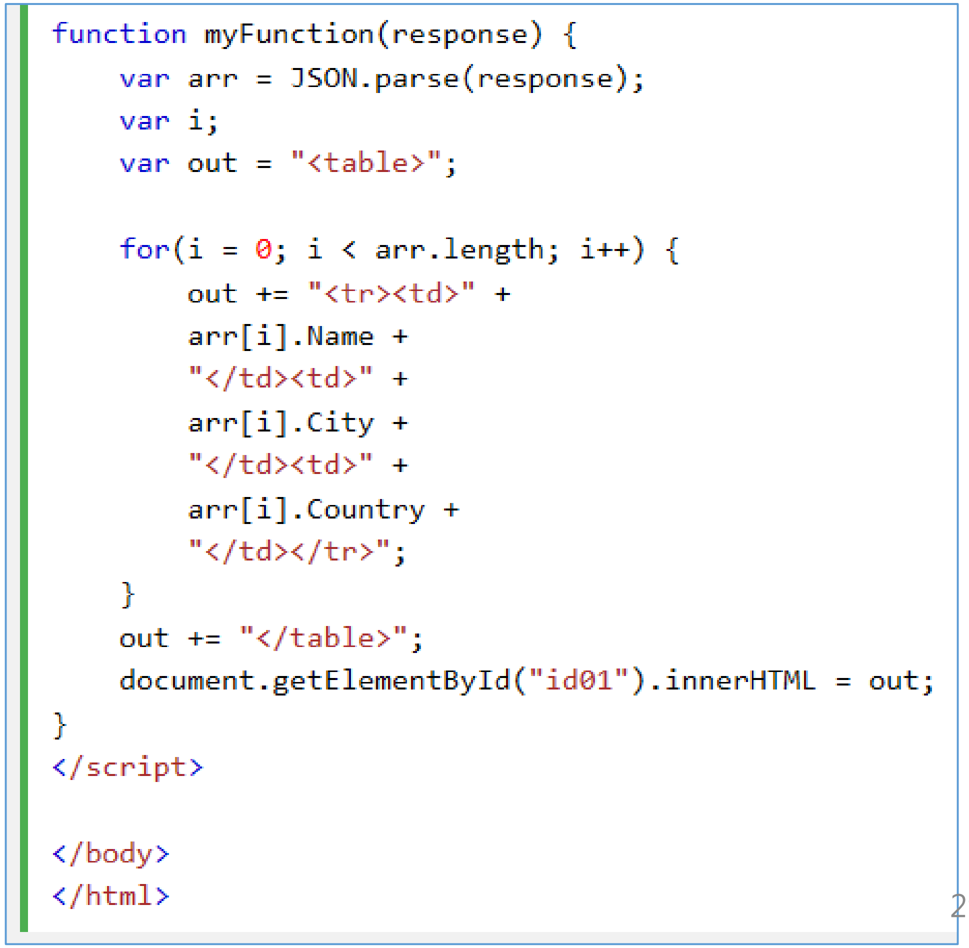
JSON SQL Example

- This example reads JSON data from a web server running PHP and MySQL:



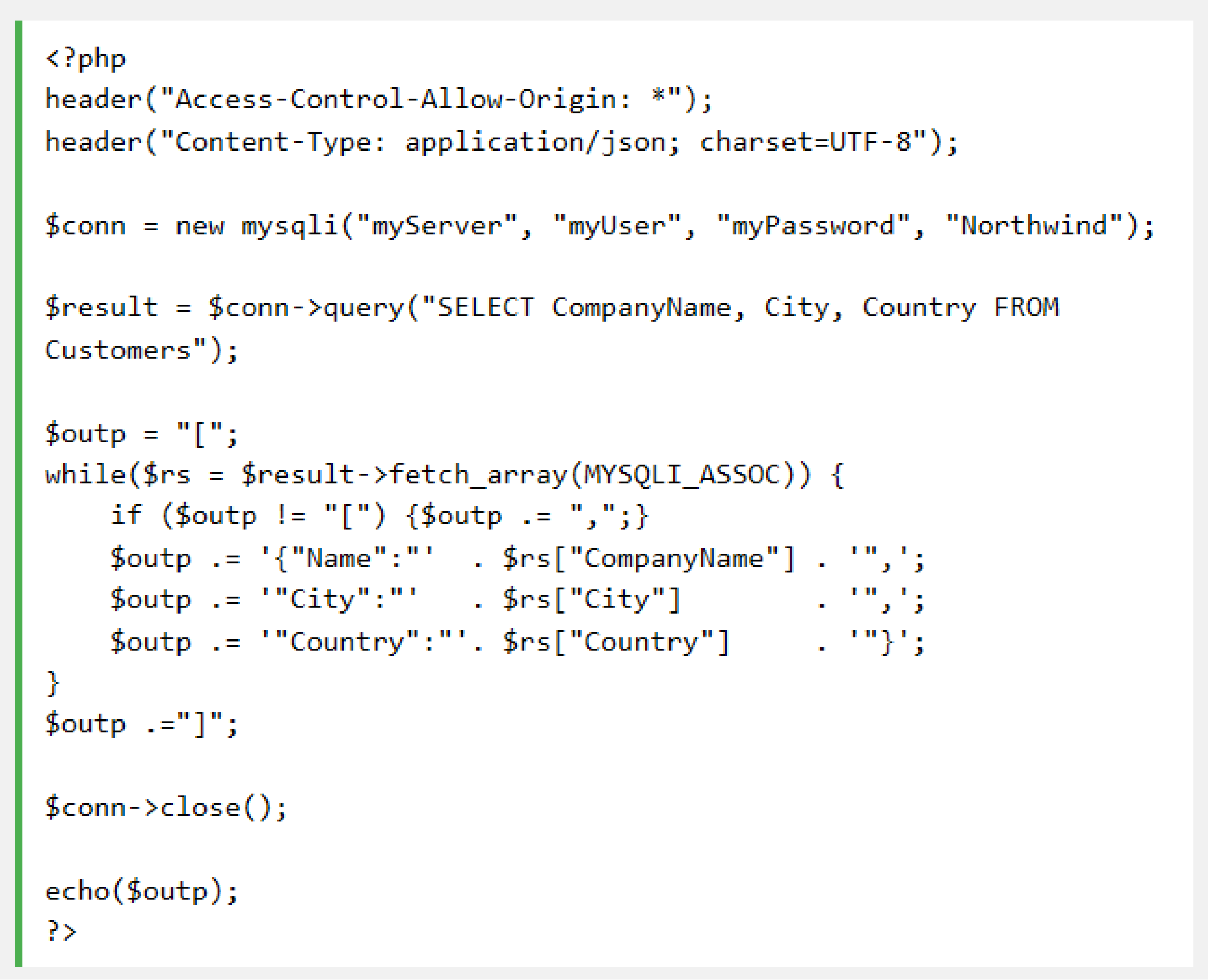
Customers.html

```
<!DOCTYPE html>
<html>
<body>

<h1>Customers</h1>
<div id="id01"></div>

<script>
var xmlhttp = new XMLHttpRequest();
var url = "http://www.w3schools.com/js/customers_mysql.php";

xmlhttp.onreadystatechange=function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this.responseText);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
```



```
function myFunction(response) {
    var arr = JSON.parse(response);
    var i;
    var out = "<table>";

    for(i = 0; i < arr.length; i++) {
        out += "<tr><td>" +
        arr[i].Name +
        "</td><td>" +
        arr[i].City +
        "</td><td>" +
        arr[i].Country +
        "</td></tr>";
    }
    out += "</table>";
    document.getElementById("id01").innerHTML = out;
}
</script>

</body>
</html>
```

The PHP Code on the Server



```
<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

$conn = new mysqli("myServer", "myUser", "myPassword", "Northwind");

$result = $conn->query("SELECT CompanyName, City, Country FROM
Customers");

$outp = "[";
while($rs = $result->fetch_array(MYSQLI_ASSOC)) {
    if ($outp != "[") {$outp .= ",";}
    $outp .= '{"Name":"'  . $rs["CompanyName"] . '",';
    $outp .= '"City":"'   . $rs["City"]        . '",';
    $outp .= '"Country":"'. $rs["Country"]     . '"}';
}
$outp .="]";

$conn->close();

echo($outp);
?>
```

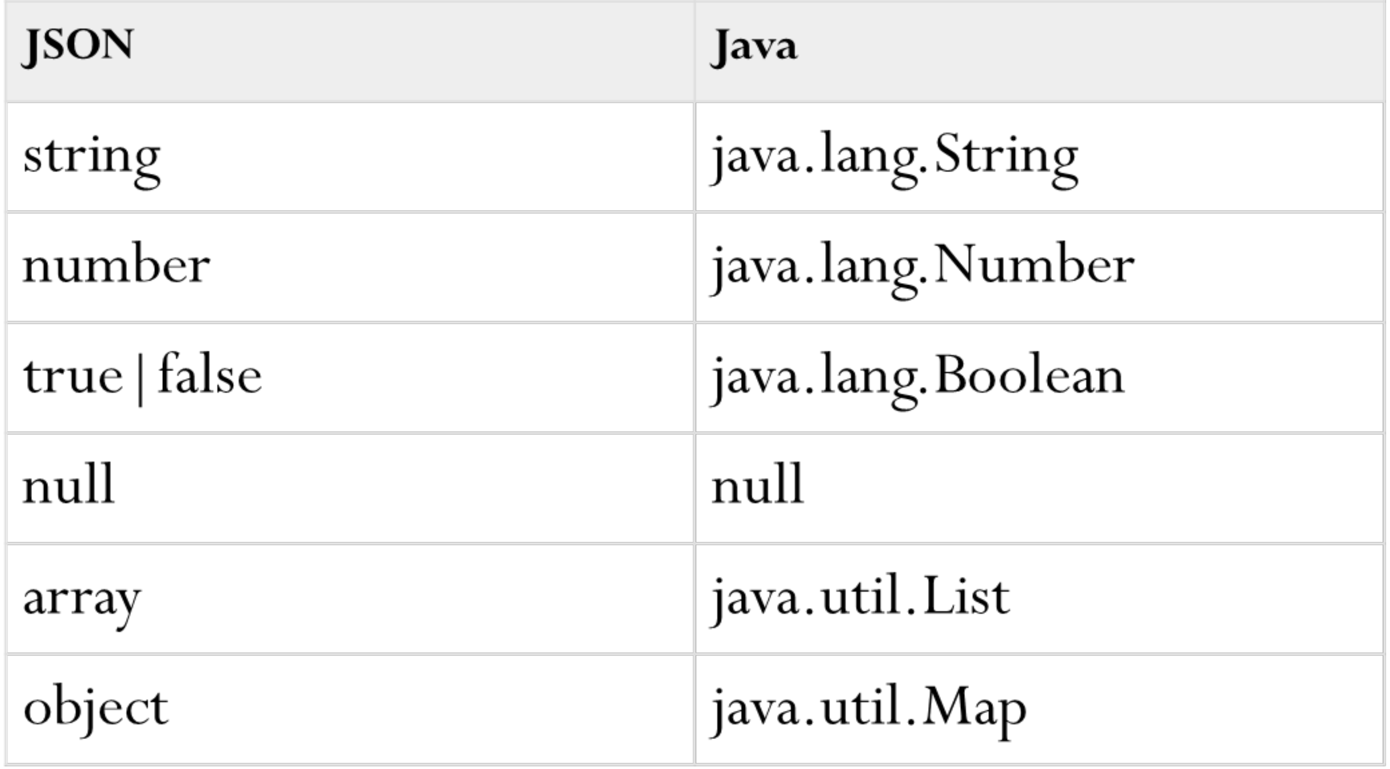
JSON and Java

Install and Environment

- Install any of the JSON modules
  - https://code.google.com/archive/p/json-simple/

- Environment variable CLASSPATH
  - Add the location of **json-simple-1.1.1.jar** file

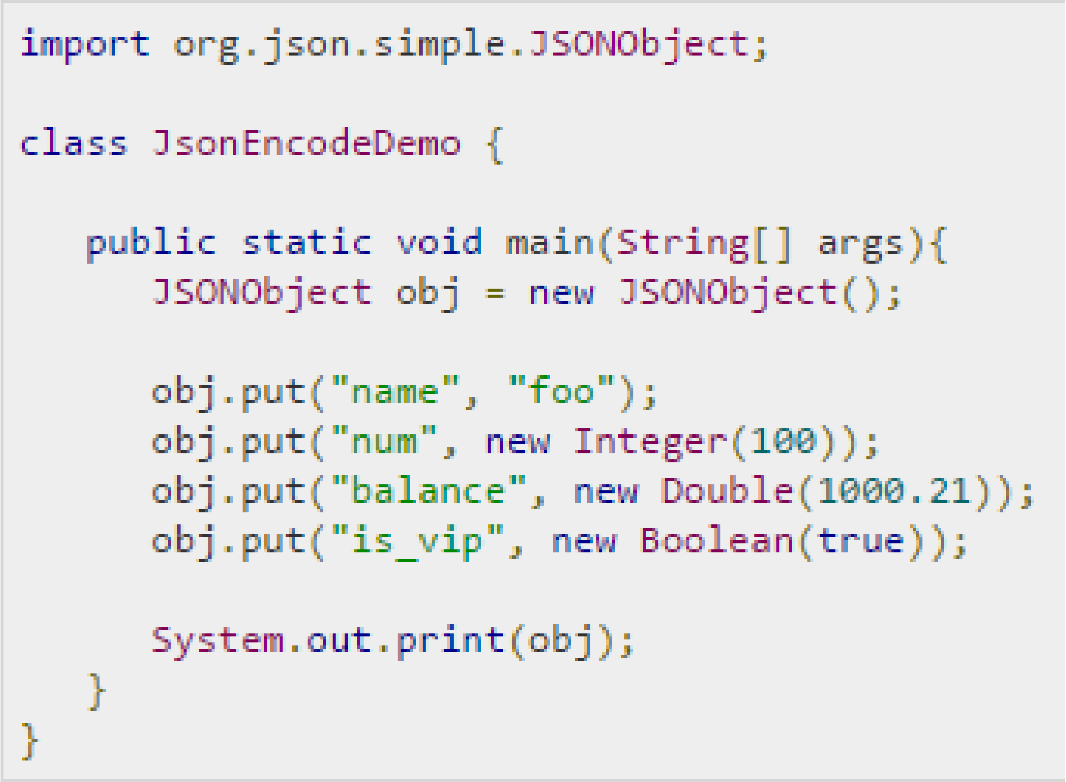
Mapping between JSON and Java



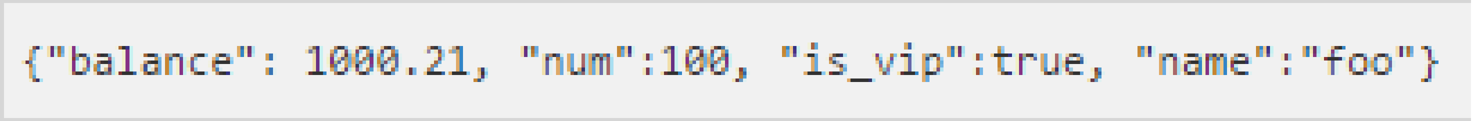
JSON	Java
string	java.lang.String
number	java.lang.Number
true|false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

- Default concrete class of *java.util.List* is *org.json.simple.JSONArray*
- Default concrete class of *java.util.Map* is *org.json.simple.JSONObject*

Encoding JSON in Java

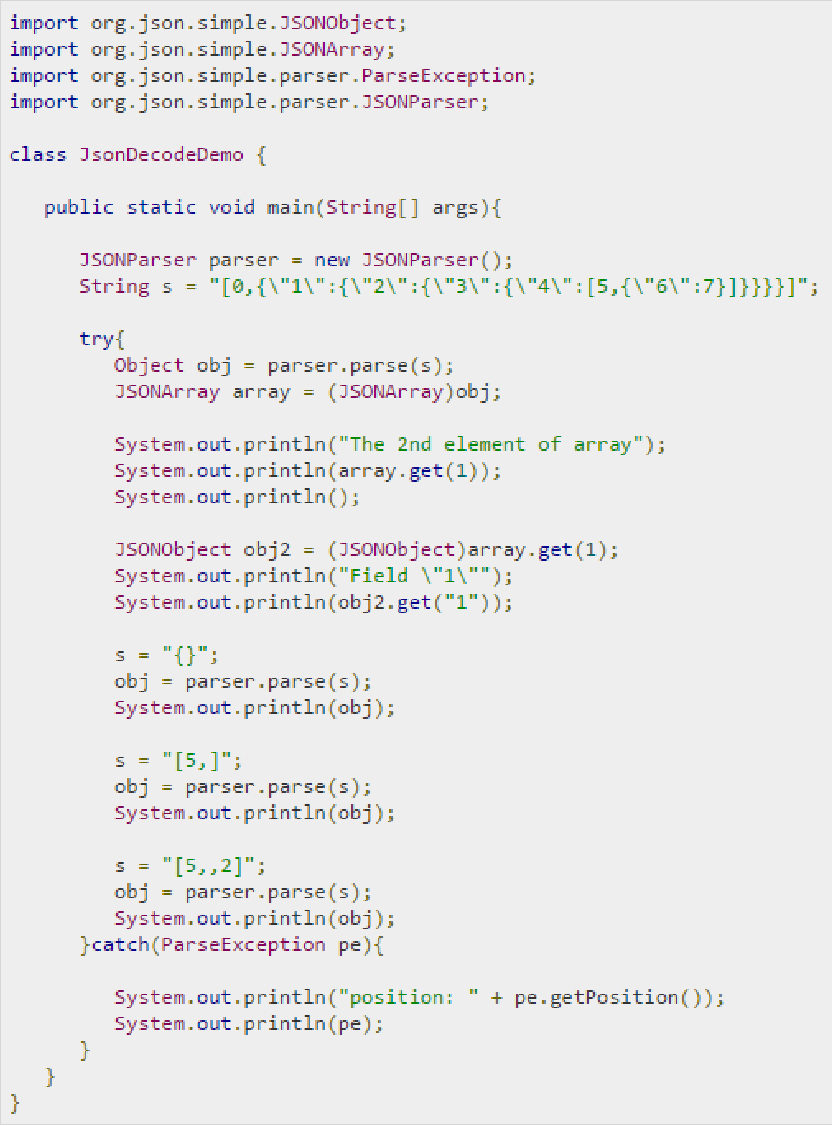


```java
import org.json.simple.JSONObject;

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);
   }
}
```



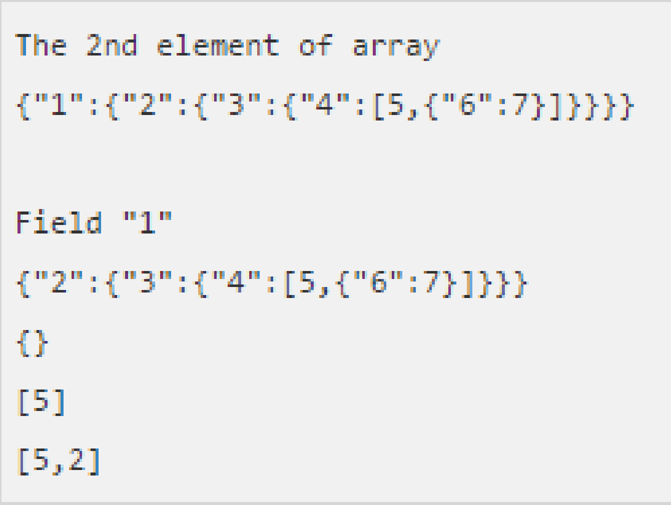
```
{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}
```

Decoding JSON in Java



```
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);
      }
   }
}
```

json은 여러가지 언어와 웹브라우저
텍스트파일 형태로 그대로 출력
복잡한 자료구조를 다른언어
왔다갔다 하면서
이용할 때 Json 매우 유용



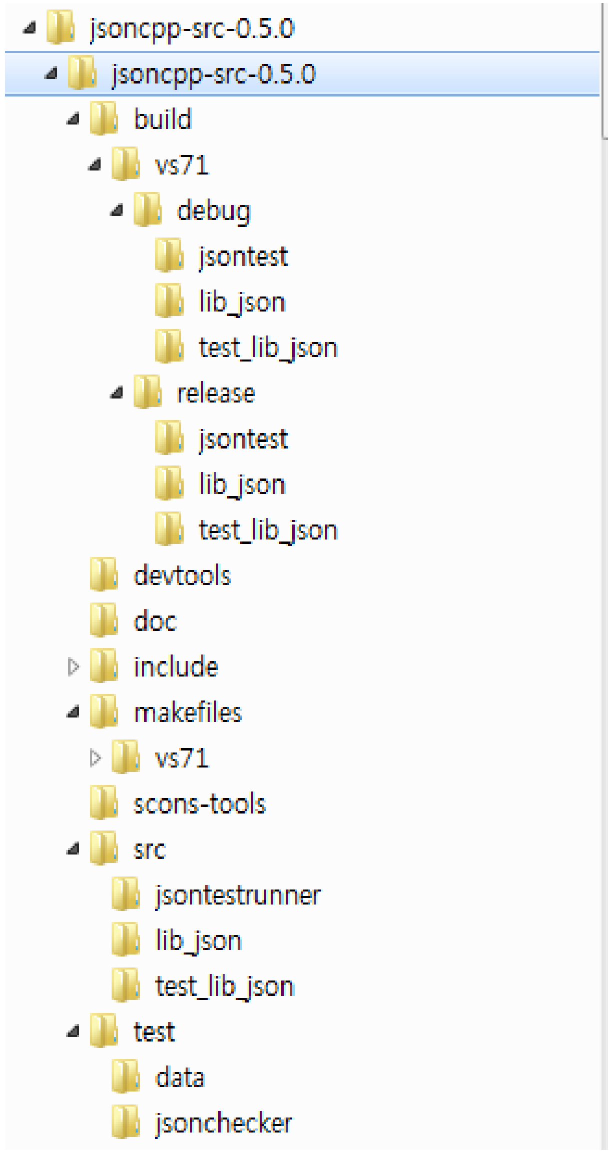
```
The 2nd element of array
{"1":{"2":{"3":{"4":[5,{"6":7}]}}}}


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

JSON C++ Library

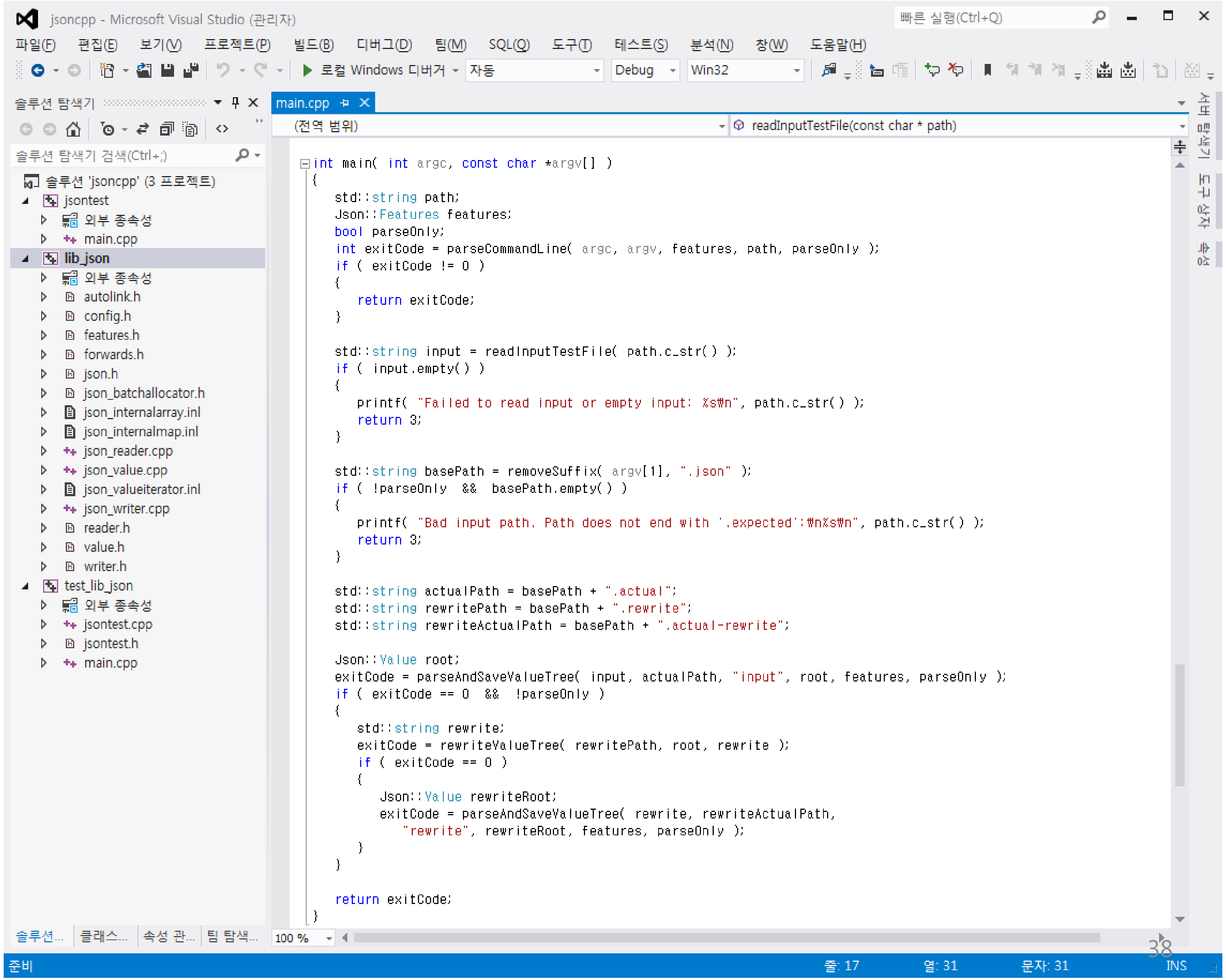
json-cpp http://sourceforge.net/projects/jsoncpp/

- json-cpp
  - http://sourceforge.net/projects/jsoncpp/
  - Download: jsoncpp-src-0.5.0.tar.gz

- 라이브러리 빌드 – Visual Studio에서
  - Jsoncpp-src-0.5.0/makefiles/vs71/jsoncpp.sln
  - Jsoncpp-src-0.5.0/build/vs71/release/lib_json/json_vc71_libmt.lib



```
jsoncpp-src-0.5.0
  jsoncpp-src-0.5.0
    build
      vs71
        debug
          jsontest
          lib_json
          test_lib_json
        release
          jsontest
          lib_json
          test_lib_json
    devtools
    doc
    include
    makefiles
      vs71
    scons-tools
    src
      jsontestrunner
      lib_json
      test_lib_json
    test
      data
      jsonchecker
```



jsoncpp - Microsoft Visual Studio (관리자)
빠른 실행(Ctrl+Q)
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) SQL(Q) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H)
로컬 Windows 디버거
자동
Debug
Win32
솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
main.cpp
(전역 범위)
readInputTestFile(const char * path)
솔루션... 클래스... 속성 관... 팀 탐색...
100 %
준비
줄: 17
열: 31
문자: 31
INS
서버 탐색기
도구 상자
속성

```
솔루션 'jsoncpp' (3 프로젝트)
  jsontest
    외부 종속성
    main.cpp
  lib_json
    외부 종속성
    autolink.h
    config.h
    features.h
    forwards.h
    json.h
    json_batchallocator.h
    json_internalarray.inl
    json_internalmap.inl
    json_reader.cpp
    json_value.cpp
    json_valueiterator.inl
    json_writer.cpp
    reader.h
    value.h
    writer.h
  test_lib_json
    외부 종속성
    jsontest.cpp
    jsontest.h
    main.cpp
```

```cpp
int main( int argc, const char *argv[] )
{
   std::string path;
   Json::Features features;
   bool parseOnly;
   int exitCode = parseCommandLine( argc, argv, features, path, parseOnly );
   if ( exitCode != 0 )
   {
      return exitCode;
   }

   std::string input = readInputTestFile( path.c_str() );
   if ( input.empty() )
   {
      printf( "Failed to read input or empty input: %s₩n", path.c_str() );
      return 3;
   }

   std::string basePath = removeSuffix( argv[1], ".json" );
   if ( !parseOnly  &&  basePath.empty() )
   {
      printf( "Bad input path. Path does not end with '.expected':₩n%s₩n", path.c_str() );
      return 3;
   }

   std::string actualPath = basePath + ".actual";
   std::string rewritePath = basePath + ".rewrite";
   std::string rewriteActualPath = basePath + ".actual-rewrite";

   Json::Value root;
   exitCode = parseAndSaveValueTree( input, actualPath, "input", root, features, parseOnly );
   if ( exitCode == 0  &&  !parseOnly )
   {
      std::string rewrite;
      exitCode = rewriteValueTree( rewritePath, root, rewrite );
      if ( exitCode == 0 )
      {
         Json::Value rewriteRoot;
         exitCode = parseAndSaveValueTree( rewrite, rewriteActualPath,
            "rewrite", rewriteRoot, features, parseOnly );
      }
   }

   return exitCode;
}
```