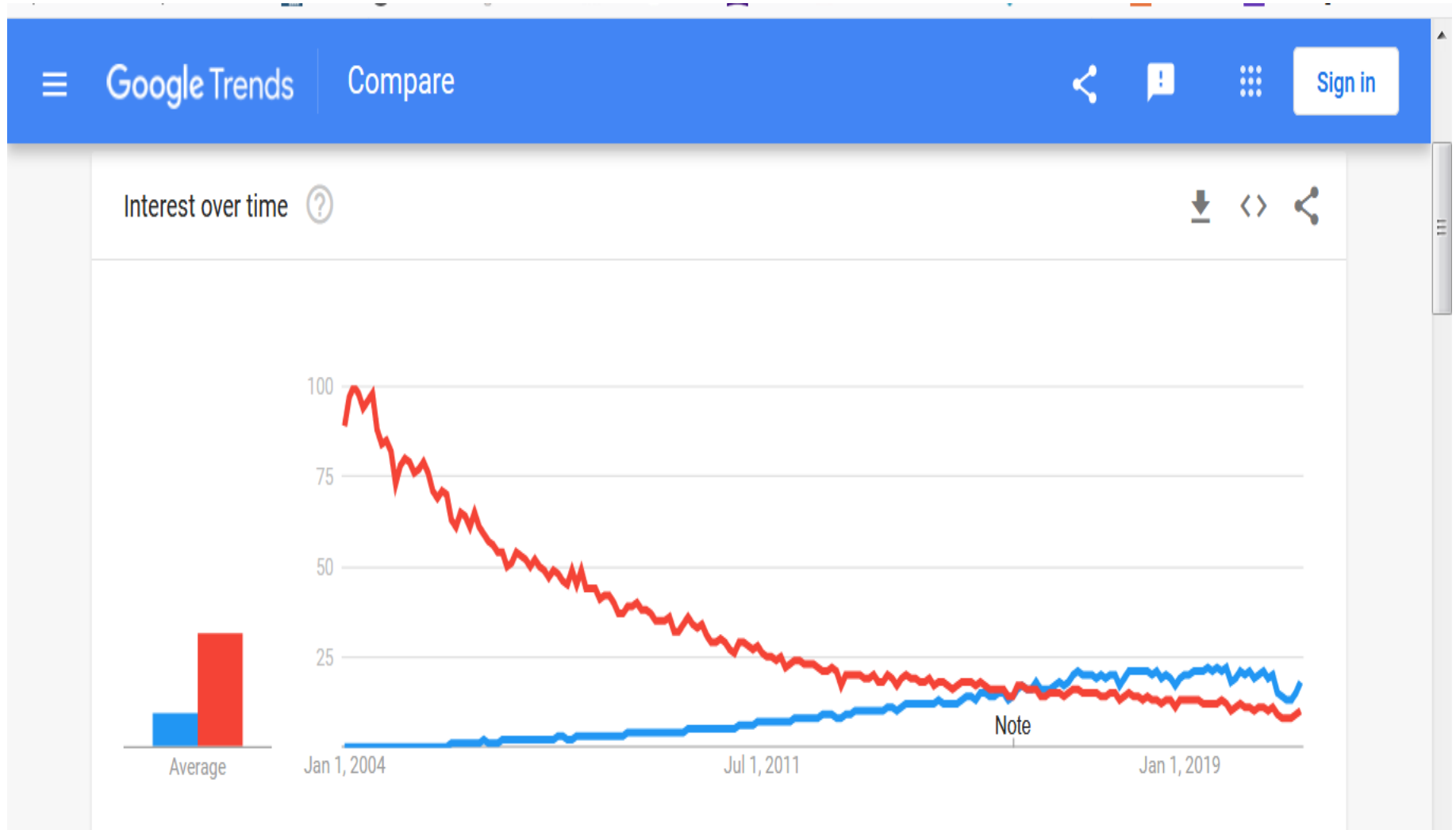


**JSON**

# What is JSON

- JSON stands for “**JavaScript Object Notation**”
  - Despite the name, JSON is a (mostly) **language-independent** way of specifying objects as name-value pairs
- Used to **format data**
- Commonly used in Web as a vehicle to **describe data being sent between systems**

# Trends in XML and JSON usage



<https://trends.google.com/trends/explore?date=all&q=JSON,XML>



Google Trends

Compare

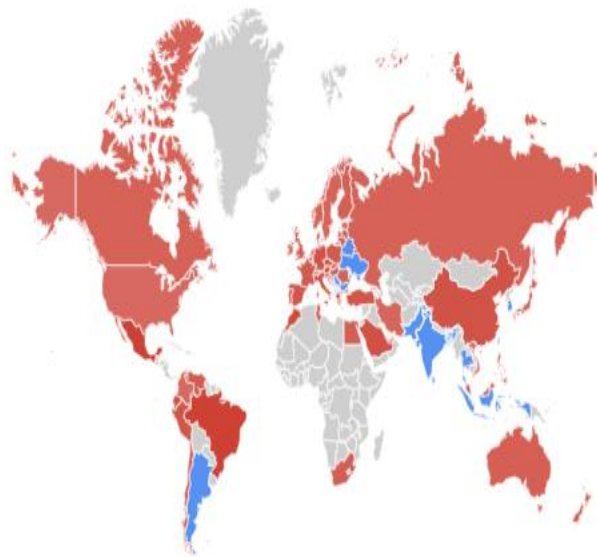


Sign in

● JSON ● XML

Worldwide, 2004 - present

● JSON ● XML



Color intensity represents percentage of searches [LEARN MORE](#)

<https://trends.google.com/trends/explore?date=all&q=json,xml>

# Specifications

**The JSON specification is:**

- <https://tools.ietf.org/html/rfc7159>

**There are two specifications for JSON-Schema**

- <http://json-schema.org/latest/json-schema-core.html>
- <http://json-schema.org/latest/json-schema-validation.html>

# JSON example

- ```
{ "skills": {  
  "web": [  
    { "name": "html",  
      "years": 5  
    },  
    { "name": "css",  
      "years": 3  
    }  
  ],  
  "database": [  
    { "name": "sql",  
      "years": 7  
    }  
  ]  
}}
```

# JSON syntax

- An *object* is an unordered set of **name/value pairs**
  - The pairs are enclosed within braces, **{ }**
  - There is a **colon** between the name and the value
  - **Pairs** are separated by **commas**
  - Example: **{ "name": "html", "years": 5 }**
- An *array* is an ordered collection of values
  - The values are enclosed within brackets **[ ]**
  - Values are separated by **commas**
  - Example: **[ "html", "xml", "css" ]**

# JSON example

- ```
{ "skills": {  
  "web": [  
    { "name": "html",  
      "years": 5  
    },  
    { "name": "css",  
      "years": 3  
    }  
  ],  
  "database": [  
    { "name": "sql",  
      "years": 7  
    }  
  ]  
}}
```



# **XML/JSON**

## **XML is a data format**

XML is a way of structuring data

## **JSON is a data format**

JSON is a way of structuring data

# Example of XML-formatted data

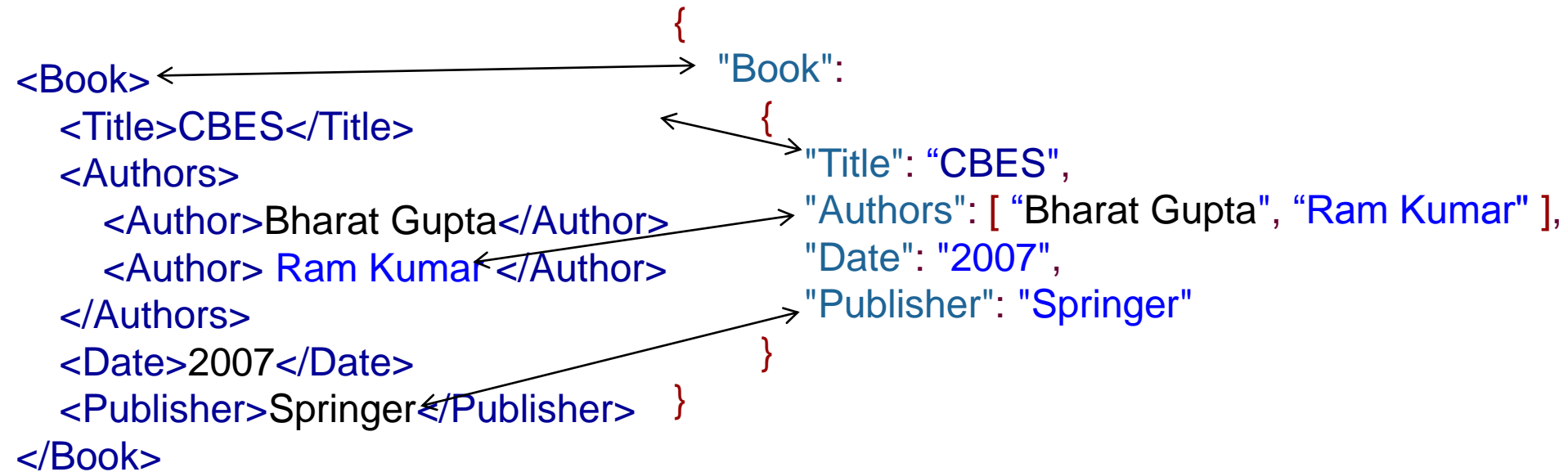
The below XML document contains data about a book: its title, authors, date of publication, and publisher.

```
<Book>
  <Title>CBES</Title>
  <Authors>
    <Author>Bharat Gupta</Author>
    <Author>Ram Kumar</Author>
  </Authors>
  <Date>2007</Date>
  <Publisher>Springer</Publisher>
</Book>
```

# Same data, JSON-formatted


```
{  
  "Book":  
    {  
      "Title": "CBES",  
      "Authors": [ "Bharat Gupta", "Ram Kumar" ],  
      "Date": "2007",  
      "Publisher": "Springer"  
    }  
}
```

# XML and JSON, side-by-side



# Creating lists in XML and JSON

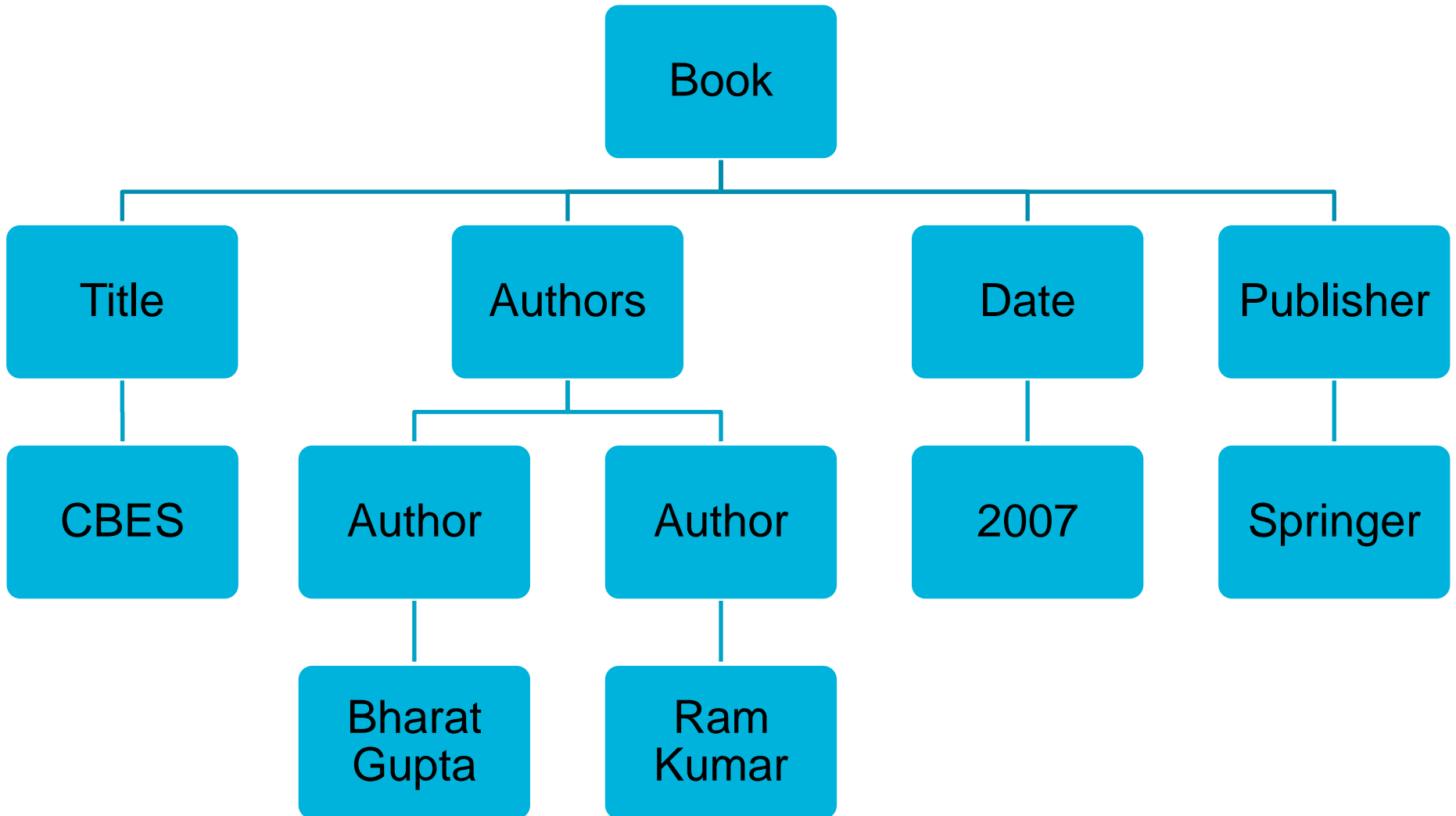
<pre>&lt;Book&gt;   &lt;Title&gt;CBES&lt;/Title&gt;   &lt;Authors&gt;     &lt;Author&gt; Bharat Gupta &lt;/Author&gt;     &lt;Author&gt; Ram Kumar   &lt;/Author&gt;   &lt;/Authors&gt;   &lt;Date&gt;2007&lt;/Date&gt;   &lt;Publisher&gt;Springer&lt;/Publisher&gt; &lt;/Book&gt;</pre>	<pre>{   "Book": {     "Title": "CBES",     "Authors": [ "Bharat Gupta", " Ram Kumar" ],     "Date": "2007",     "Publisher": "Springer"   } }</pre>
---	--



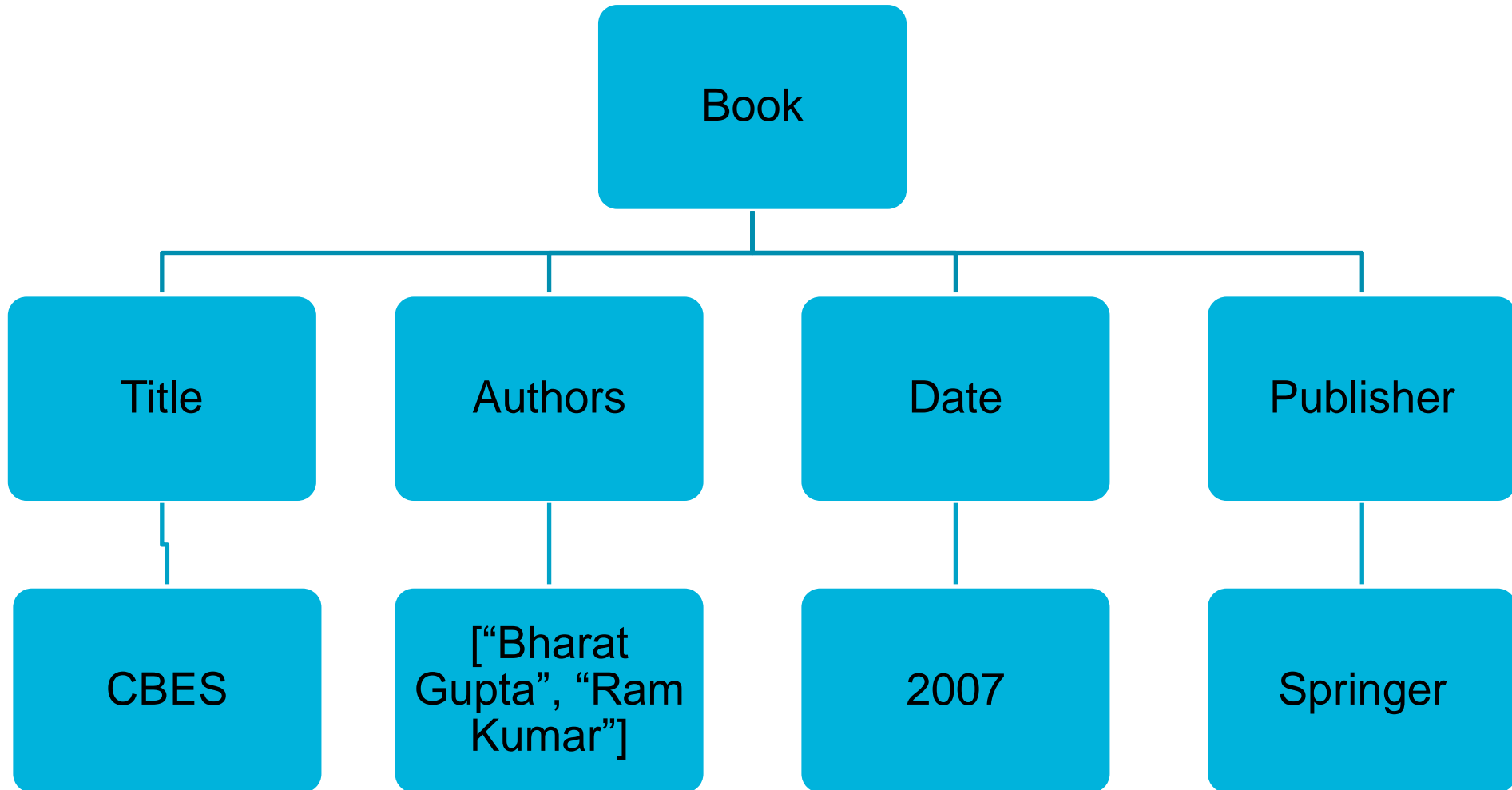
# **XML/JSON is a meta-language**

- **XML/JSON is a language that you use to create other languages.**
- **For example, XML/JSON is used to create a Book language, consisting of <Book>, <Title>, <Author>, and so forth.**

# A XML document is a tree



# A JSON Object is a tree





# XML Schema for Book

```
<xs:element name="Book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Title" type="xs:string" />
      <xs:element name="Authors">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Author" type="xs:string" maxOccurs="5"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Date" type="xs:gYear" />
      <xs:element name="Publisher" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Springer" />
            <xs:enumeration value="MIT Press" />
            <xs:enumeration value="Harvard Press" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Equivalent JSON Schema

```
{  
  "$schema": http://json-schema.org/draft-04/schema,
```

```
  "type": "object",
```

```
  "properties": {
```

```
    "Book": {
```

```
      "type": "object",
```

```
      "properties": {
```

```
        "Title": {"type": "string"},
```

```
        "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
```

```
        "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
```

```
        "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
```

```
      },
```

```
      "required": ["Title", "Authors", "Date"],
```

```
      "additionalProperties": false
```

```
    }
```

```
  },
```

```
  "required": ["Book"],
```

```
  "additionalProperties": false
```

```
}
```

```
<xs:element name="Book">
```

```
  <xs:complexType>
```

```
    <xs:sequence>
```

```
      <xs:element name="Title" type="xs:string" />
```

```
      <xs:element name="Authors">
```

```
        <xs:complexType>
```

```
          <xs:sequence>
```

```
            <xs:element name="Author" type="xs:string" maxOccurs="5"/>
```

```
          </xs:sequence>
```

```
        </xs:complexType>
```

```
      </xs:element>
```

```
      <xs:element name="Date" type="xs:gYear" />
```

```
      <xs:element name="Publisher" minOccurs="0">
```

```
        <xs:simpleType>
```

```
          <xs:restriction base="xs:string">
```

```
            <xs:enumeration value="Springer" />
```

```
            <xs:enumeration value="MIT Press" />
```

```
            <xs:enumeration value="Harvard Press" />
```

```
          </xs:restriction>
```

```
        </xs:simpleType>
```

```
      </xs:element>
```

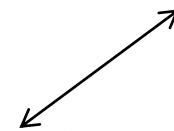
```
    </xs:sequence>
```

```
  </xs:complexType>
```

```
</xs:element>
```

# Title with string type

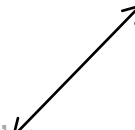
```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "Book": {
      "type": "object",
      "properties": {
        "Title": {"type": "string"},
        "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
        "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
        "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
      },
      "required": ["Title", "Authors", "Date"],
      "additionalProperties": false
    }
  },
  "required": ["Book"],
  "additionalProperties": false
}
```



The diagram shows an arrow pointing from the JSON Schema property `"Title": {"type": "string"}` to its corresponding XML Schema representation: `<xs:element name="Title" type="xs:string" />`.

# Authors list

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "Book": {
      "type": "object",
      "properties": {
        "Title": {"type": "string"},
        "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
        "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
        "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
      },
      "required": ["Title", "Authors", "Date"],
      "additionalProperties": false
    }
  },
  "required": ["Book"],
  "additionalProperties": false
}
```



The diagram illustrates the mapping between a JSON Schema property and its XML Schema definition. An arrow points from the **"Authors"** property in the JSON Schema to the corresponding XML Schema definition for the **Authors** element.

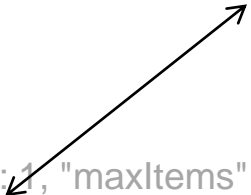
**XML Schema Definition:**

```
<xs:element name="Authors">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Author" type="xs:string" maxOccurs="5"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Date with year type

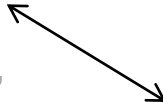
```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "Book": {
      "type": "object",
      "properties": {
        "Title": {"type": "string"},
        "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
        "Date": {"type": "string", "pattern": "[0-9]{4}"},
        "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
      },
      "required": ["Title", "Authors", "Date"],
      "additionalProperties": false
    }
  },
  "required": ["Book"],
  "additionalProperties": false
}
```

`<xs:element name="Date" type="xs:gYear" />`



# Publisher with enumeration

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "Book": {
      "type": "object",
      "properties": {
        "Title": {"type": "string"},
        "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
        "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
        "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
      },
      "required": ["Title", "Authors", "Date"],
      "additionalProperties": false
    }
  },
  "required": ["Book"],
  "additionalProperties": false
}
```

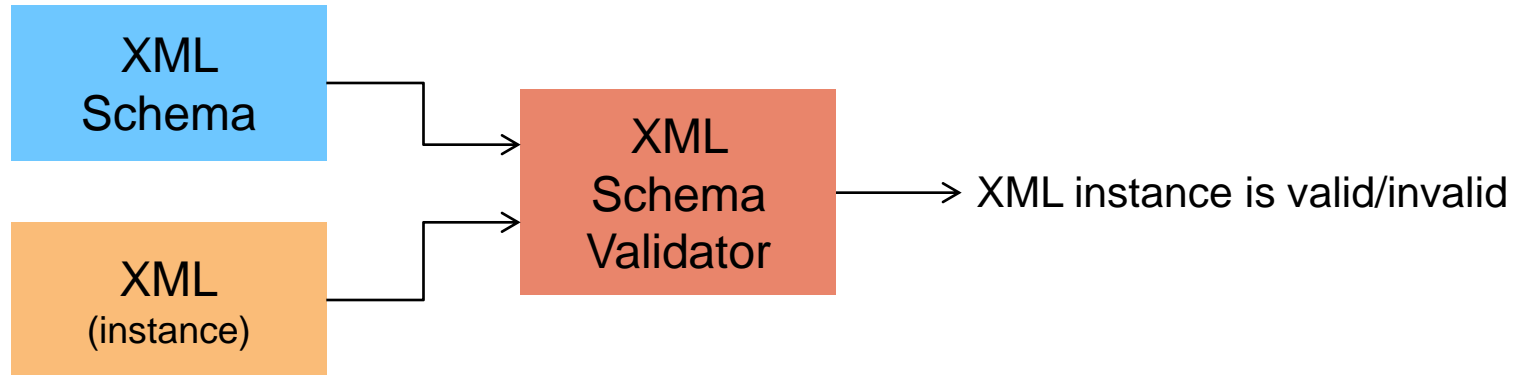


```
<xs:element name="Publisher" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Springer" />
      <xs:enumeration value="MIT Press" />
      <xs:enumeration value="Harvard Press" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Schema language

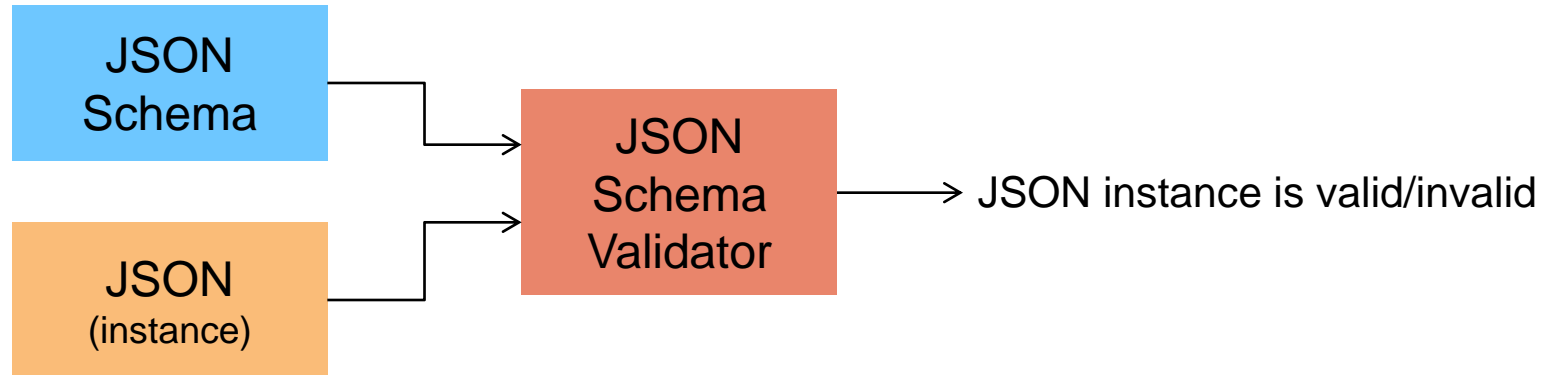
- **An XML Schema is written in XML.**
- **A JSON Schema is written in JSON.**

# Validate XML docs against XML Schema





# Validate JSON docs against JSON Schema



# **JSON Schema validators**

**<http://json-schema.org/implementations.html>**

# Online JSON Schema validator

<http://json-schema-validator.herokuapp.com/index.jsp>

Schema:

Paste your JSON Schema in here 1

Data:

Paste your JSON in here 2

Validate [\(load sample data\)](#)

Click on the validate button 3

Validation results:

Results of validation is shown here 4

# Status of JSON Schema specifications

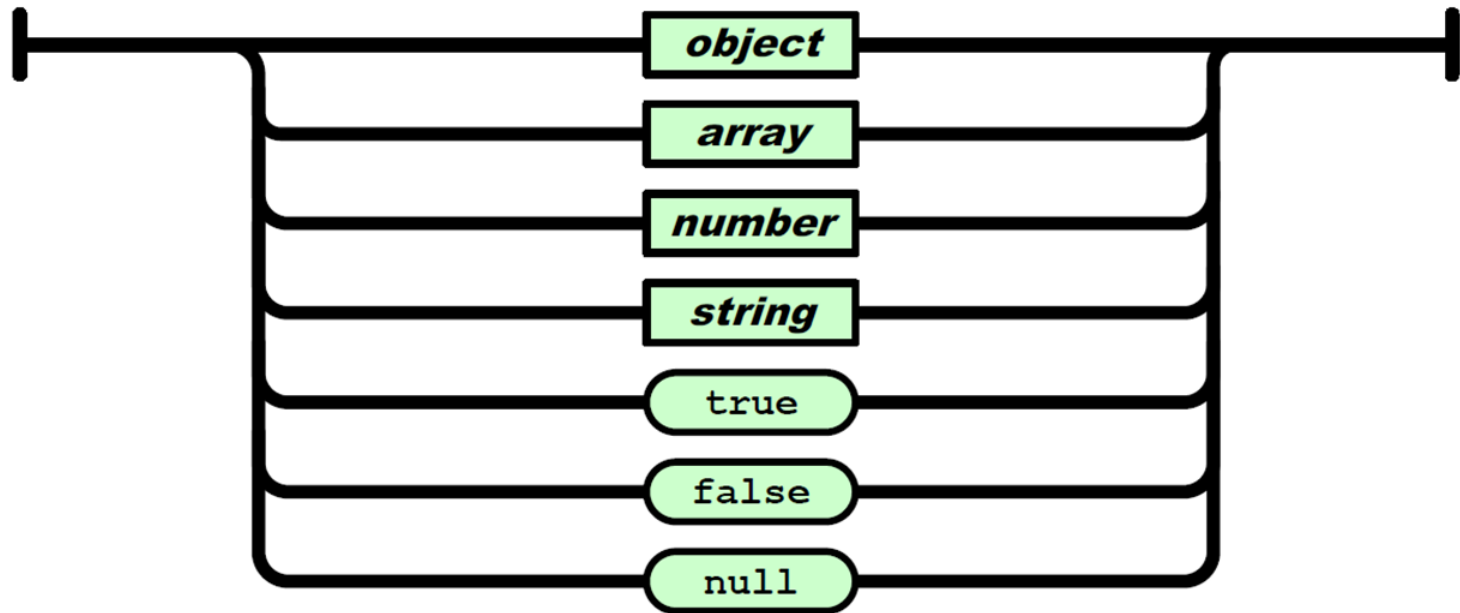
- JSON Schema is being developed under the IETF standards organization.
- The JSON Schema specification is currently at draft #4.
- Work is proceeding on draft #5.

# **The JSON data format**

# JSON value

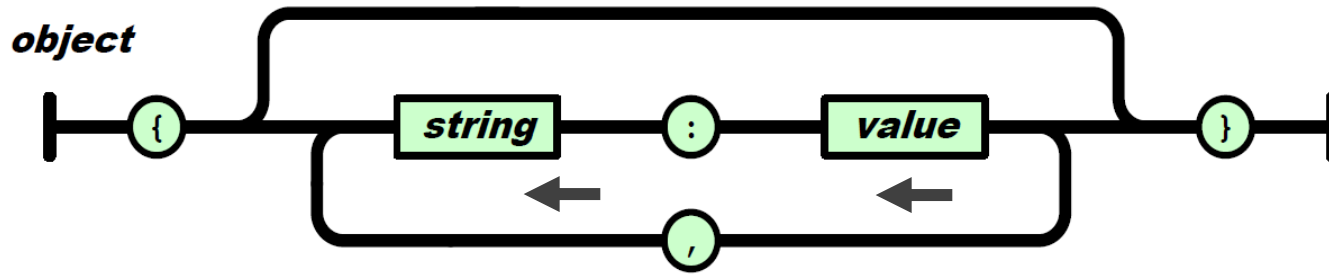
A JSON instance contains a single JSON value. A JSON **value** may be either an **object**, **array**, **number**, **string**, **true**, **false**, or **null**

*value*



# JSON object

A JSON object is **zero or more *string-colon-value* pairs**, separated by comma and wrapped within curly braces:

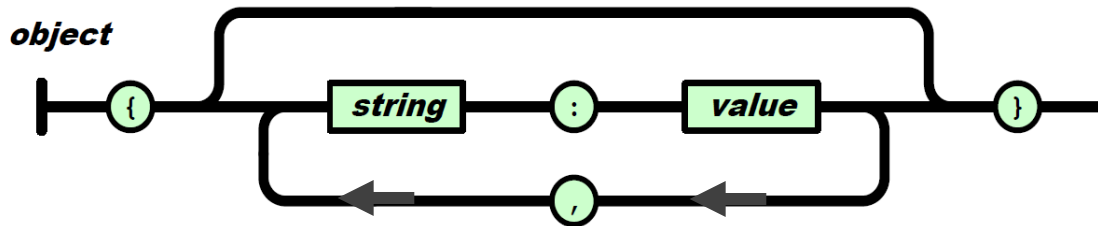


Example of a JSON object:

```
{ "name": "John Doe",  
  "age": 30,  
  "married": true }
```

# Empty object

A JSON object **may be empty**.



This is a JSON object: { }



# No duplicate keys

- You should consider JSON objects as containing key/value pairs.
- Just as in a database the primary keys must be **unique**, so a **JSON object the keys must be unique**.
- This JSON object has duplicate keys:

```
{ "Title": "A story by Mark Twain",  
  "Title": "The Adventures of Huckleberry Finn" }
```

# JSON parsers have unpredictable behavior on JSON objects with duplicate keys

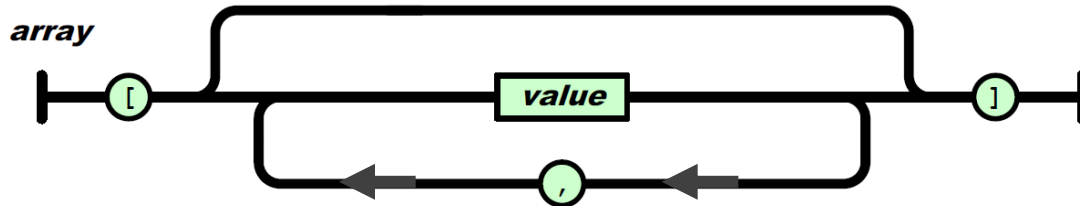
A JSON object whose names are all unique is interoperable in the sense that all software implementations receiving that object will agree on the name-value mappings. When the names within an object are not unique, the behavior of software that receives such an object is unpredictable. Many implementations report the last name/value pair only. Other implementations report an error or fail to parse the object and some implementations report all of the name/value pairs, including duplicates.

RFC 7159

# JSON array

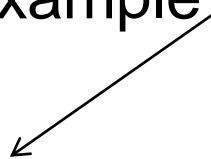
A JSON array is used to express a list of values.

A JSON array contains zero or more values, separated by comma and **wrapped within square brackets**:



```
{ "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": ["John", "Mary", "Pat"] }
```

Example of a JSON array



# Empty array vs. array with a null value

[ ]



Array with no items in it.

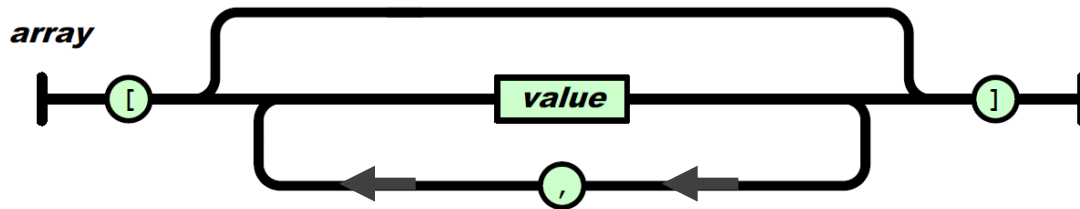
[ null ]



Array with one item in it.

# Array of objects

Each item in an array may be any of the **seven JSON values**.



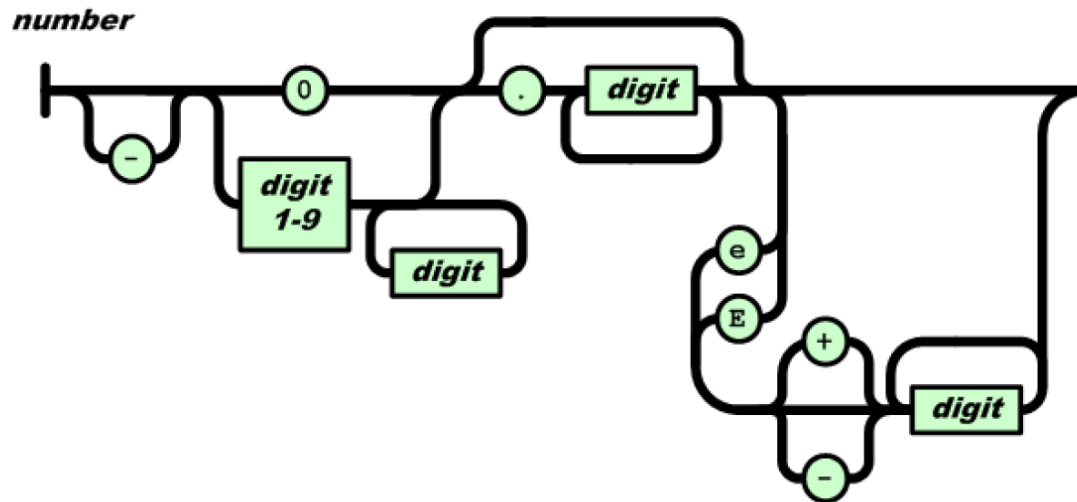
```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": [  
    {"name": "John", "age": 25},  
    true, "Hello World"  
  ]  
}
```

The array contains 3 items.  
**The first item is an object,**  
the second item is a boolean,  
and **the third item is a string.**



# JSON number

A number is an integer or a decimal and it may have an exponent:



```
{  
  "name": "John Doe",  
  "age": 30, ← Example of a JSON number  
  "married": true,  
  "siblings": ["John", "Mary", "Pat"]  
}
```

# JSON string

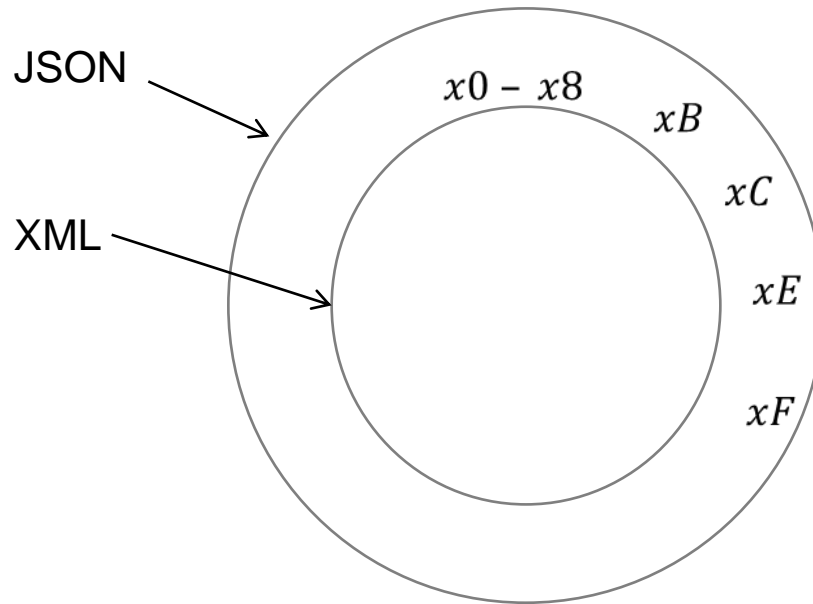
**A string is a sequence of Unicode characters wrapped within quotes (").**

```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": ["John", "Mary", "Pat"]  
}
```

← Example of a JSON string



# JSON chars are a superset of XML chars



**Be careful converting JSON to XML as the result may be a non-well-formed XML document.**

# These characters must be escaped

If any of the following characters occur within a string, they must be *escaped* by preceding them with a **backslash (\)**:

- quotation mark ("),
- backslash (\),
- the control characters U+0000 to U+001F

# Each character corresponds to a number

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

ASCII table

# Expressing characters in hex format

A character may be represented as a hexadecimal number using this notation: `\uXXXX`

For example: `\u006A` or `\u006a`.

# Unicode details

- Use the `\uXXXX` notation for Unicode code points in the Basic Multilingual Plane.

# No multiline strings

JSON does not allow multiline strings.

Legal:

```
{  
  "comment": "This is a very, very long comment"  
}
```

Not legal:

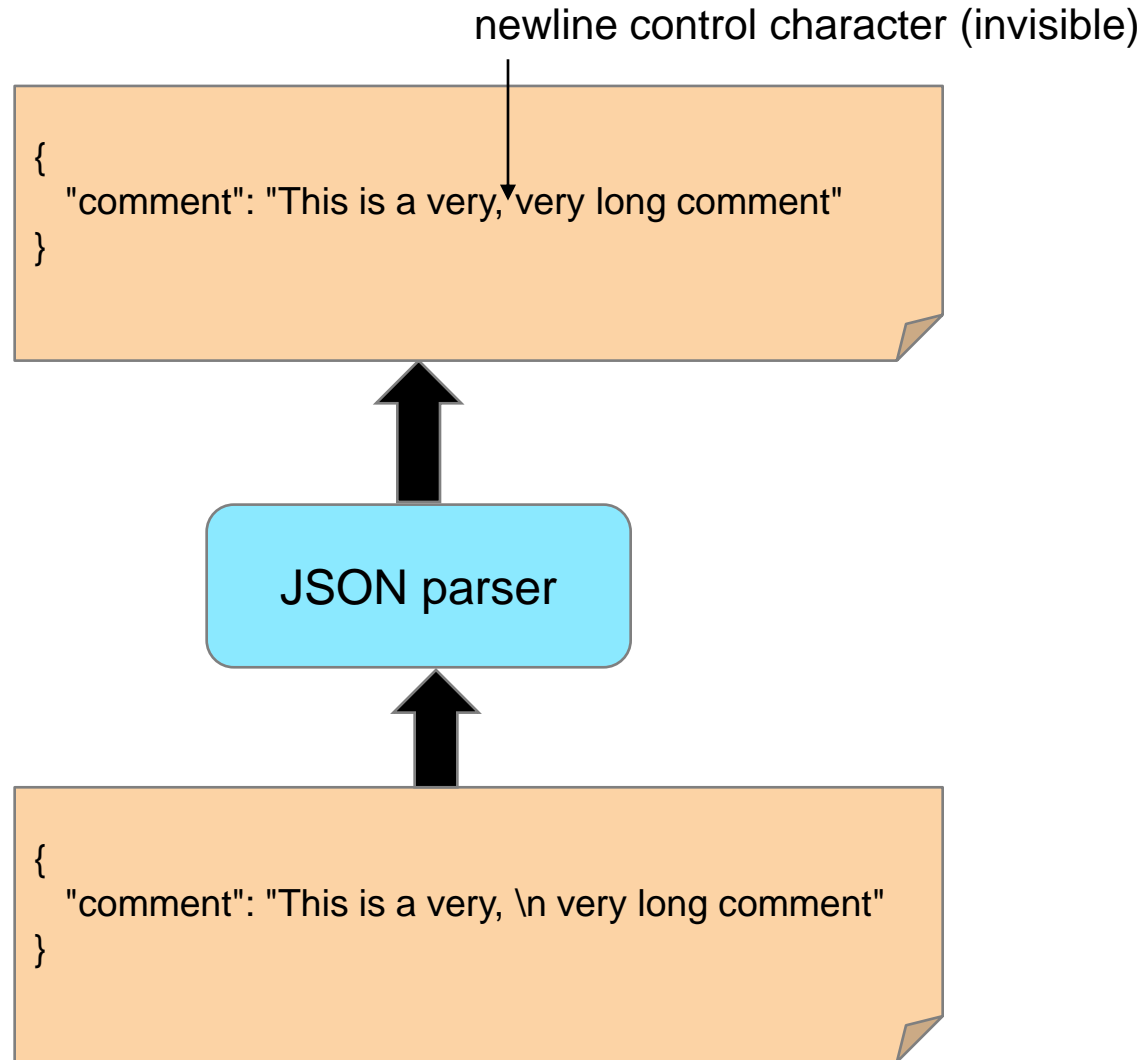
```
{  
  "comment": "This is a very,  
  very long comment"  
}
```

# No multiline strings (cont.)

Legal:

```
{  
  "comment": "This is a very, \n very long comment"  
}
```

## JSON parser converts \n to the newline character





# Achieving interoperability in a world where different OS's represent newline differently

Each operating system has its own convention for signifying the end of a line of text:

Unix: the newline is a character, with the value hex A (LF).

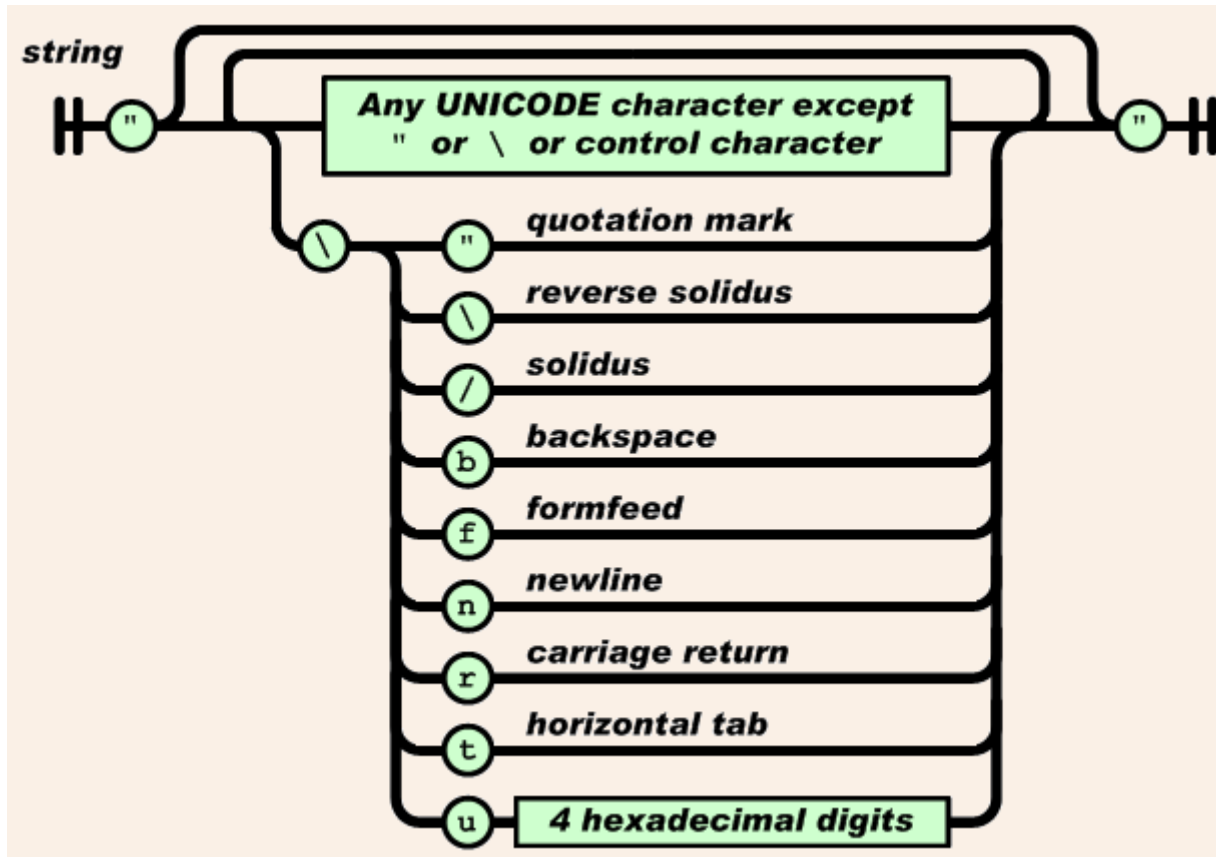
MS Windows: the newline is a combination of two characters, with the values hex D (CR) and hex A (LF), in that order.

Mac OS: the newline is a character, with the value hex D (CR).

This operating-system-dependency of newlines can cause interoperability problems, e.g., the newlines in a string created on a Unix box will not be understood by applications running on a Windows box.

- The newline problem is resolved in XML and in JSON:
- XML: all newlines are normalized by an XML parser to hex A (LF). So it doesn't matter whether you create your XML document on a Unix box, a Windows box, or a Macintosh box, all newlines will be represented as hex A (LF).
- JSON: multi-line strings are not permitted! So the newline problem is avoided completely. You can, however, embed within your JSON strings the `\n` (LF) or `\r\n` (CRLF) symbols, to instruct processing applications: "Hey, I would like a newline here."

# JSON strings



<http://json.org/>

# Other JSON values

The values **true**, **false**, and **null** are literal values; they are **not wrapped in quotes**.

```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": ["John", "Mary", "Pat"]  
}
```

← Example of a JSON boolean

# `null`

**Some people do not have a middle name, we can use `null` to indicate “no value”:**

```
{  
  "first-name": "John",  
  "middle-name": null,  
  "last-name": "Doe"  
}
```

# Legal JSON instance

42

“Hello World”

True

[ true, null, 12, "ABC" ]

# Whitespace is irrelevant

```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true  
}
```



***equivalent***

```
{"name":"John Doe","age":30,"married":true}
```

# String delimiters: JSON vs. XML

- JSON strings are always delimited by **double quotes**.
- XML strings (such as attribute values) may be delimited by **either double quotes or single quotes**.



# Using JSON you can define arbitrarily complex structures

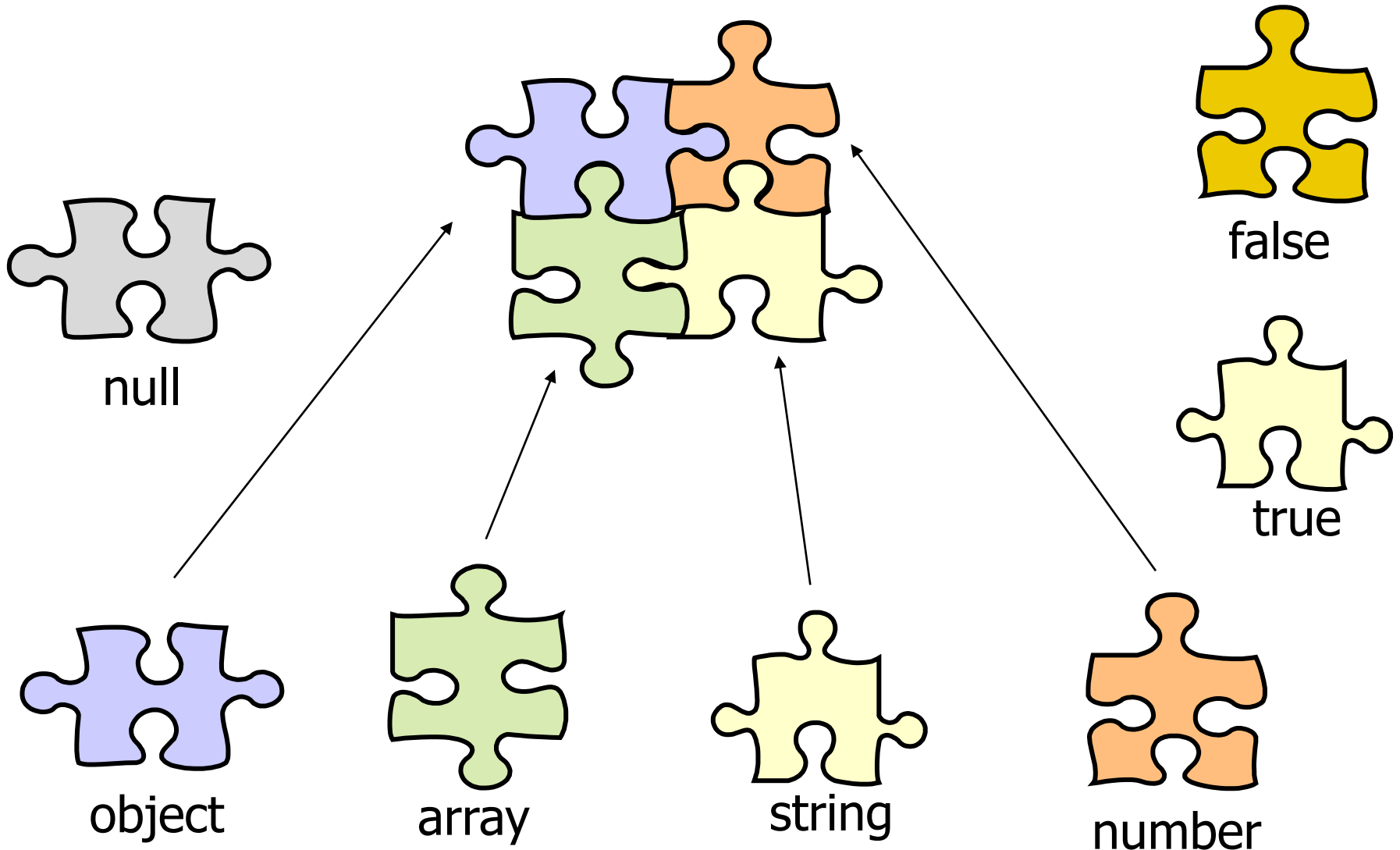
```
{
  "Book":
  {
    "Title": "Parsing Techniques",
    "Authors": [ "Dink Grune", "Ceriél J.H. Jacobs" ]
  }
}
```

```
{
  "Book":
  {
    "Title": "Parsing Techniques",
    "Authors": [
      {"name": "Dink Grune", "university": "ABC"},
      {"name": "Ceriél J.H. Jacobs", "university": "XYZ"}
    ]
  }
}
```

# Extend

```
{  
  "Book":  
    {  
      "Title": "Parsing Techniques",  
      "Authors": [  
        {"name": {"first": "Dink", "last": "Grune"}, "university": "ABC"},  
        {"name": {"first": "Ceriél", "last": "Jacobs"}, "university": "XYZ"}  
      ]  
    }  
}
```

# 7 simple JSON components



# JSON provides the structures and assembly points, you customize them for your needs

object

```
{  
  "___": json-value,  
  "___": json-value,  
  "___": json-value,  
  ...  
}
```

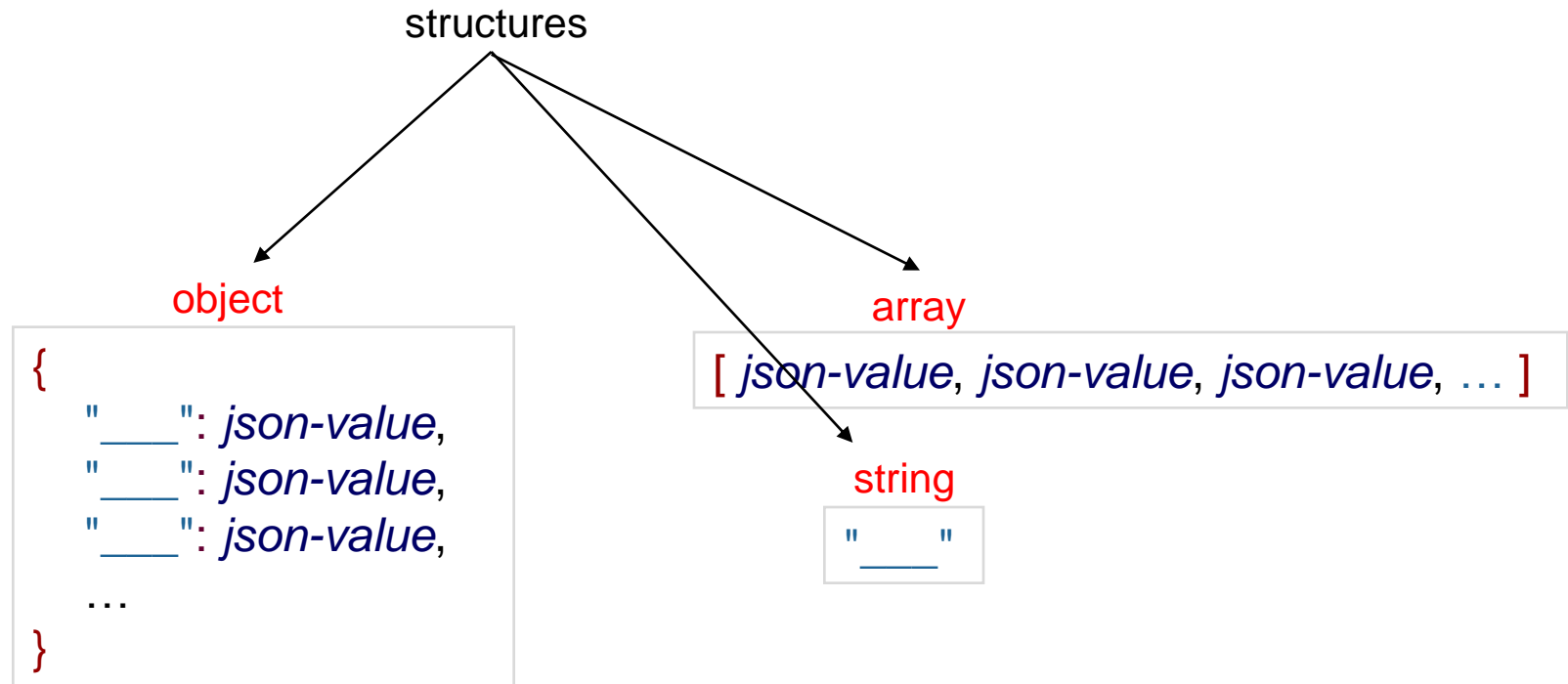
array

```
[ json-value, json-value, json-value, ... ]
```

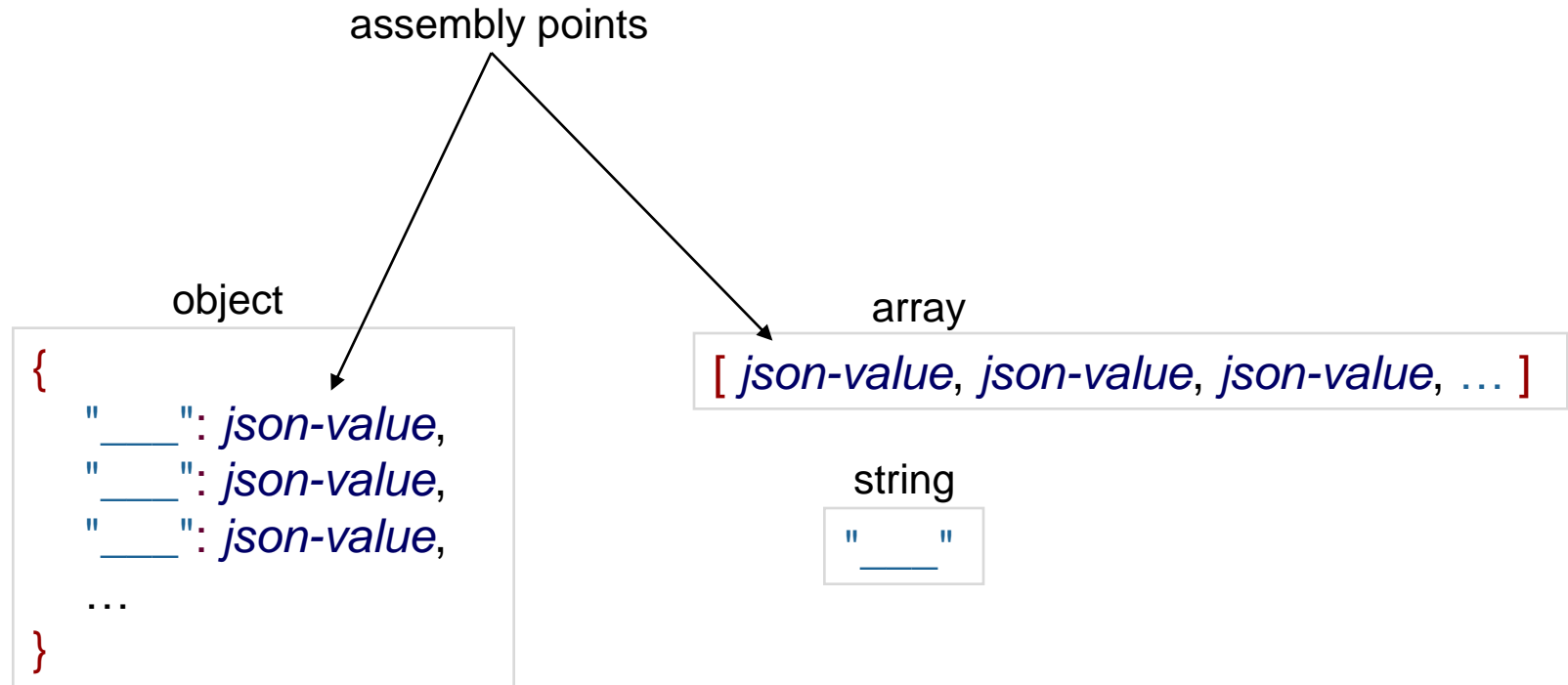
string

```
"___"
```

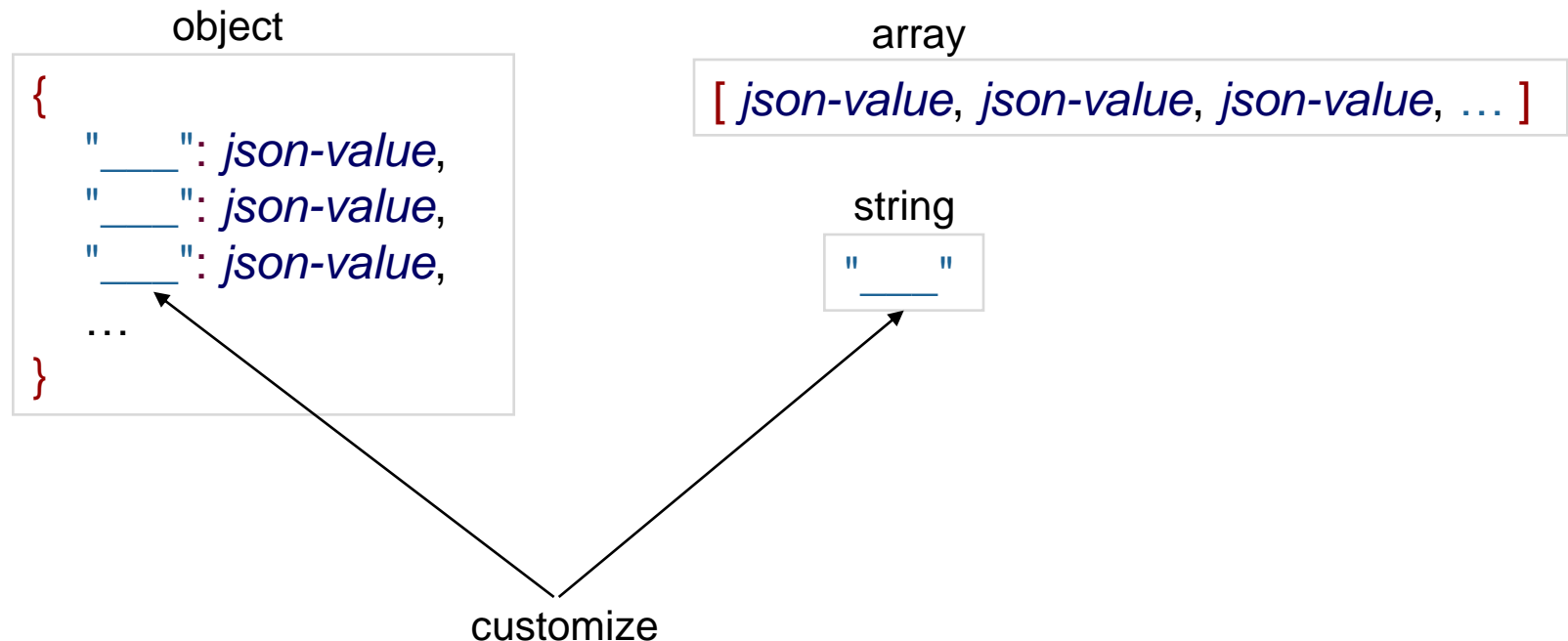
# JSON provides the structures and assembly points, you customize them for your needs



# JSON provides the structures and assembly points, you customize them for your needs



# JSON provides the structures and assembly points, you customize them for your needs



# Comments not allowed

- You cannot comment a JSON instance document.
- There is no syntax for commenting JSON instances.



# JSON

- There are **no attributes** in JSON.
- **No namespaces** in JSON.
- There is no equivalent of XSLT in JSON. In JSON text documents should be processed by a general purpose language such as Java or JavaScript, not a domain-specific language such as XSLT.
- JSON to XML should be straightforward, except when **JSON contains control characters** (which are not allowed in XML).
- From XML → JSON, attributes and namespaces has to be handled.

# JSONx (JSON → XML)

**JSONx is an IBM standard format to represent JSON as XML**

This document specifies a mapping between JSON and XML, known as JSONx. It is used by several IBM products.

<http://tools.ietf.org/html/draft-rsalz-jsonx-00>

```
"phoneNumbers": [  
  "212 555-1111",  
  "212 555-2222"  
]
```



```
<json:array name="phoneNumbers">  
  <json:string>212 555-1111</json:string>  
  <json:string>212 555-2222</json:string>  
</json:array>
```

# **Creating JSON Schemas**

# MS Visual Studio supports JSON Schema

- Visual Studio provides a editor that helps you create JSON Schemas.
- The editor is available in the free Community version of Visual Studio 2013.
- How to create JSON Schemas using Visual Studio:  
<https://www.youtube.com/watch?v=Jt5SCNC87d4>

# **A contract for data exchanges**

**Both XML Schema and JSON Schema may be used as a contract for data exchanges:**

# Fundamental difference between XML Schema and JSON Schema

- **XML Schema**: specifies *closed content* unless deliberate measures are taken to make it open (e.g., use of <any> element liberally throughout the schema).
- **JSON Schema**: specifies *open content* unless deliberate measures are taken to make it closed (e.g., use of "additionalProperties": false liberally throughout the schema).

**Open Content**: instance documents can contain items above and beyond those specified by the schema.

**Closed Content**: instance documents can contain only those items specified by the schema.

# JSON Schema

- **Regular expressions (regexes):** JSON Schema uses regexes a lot. The regular expression language is well-established and very powerful. Most data constraints can be expressed using regexes.
- **Recursion:** nearly all keywords in JSON Schema are recursively defined.
- **Simplicity:** JSON Schema doesn't have a large set of data types or other features. It employs simple components which may be assembled to generate arbitrary complexity.

# A JSON Schema is a JSON object

```
{  
  "keyword": value,  
  "keyword": value,  
  "keyword": value,  
  ...  
}
```

The JSON Schema specification defines the set of keywords and values that can be used to construct a schema.

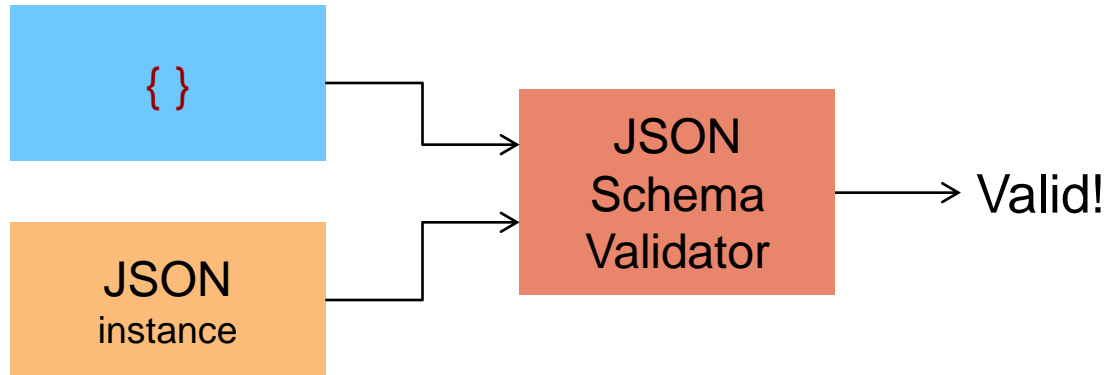


# Empty schema



This is a valid JSON Schema.  
It places no restrictions on  
JSON instances.

# Every JSON instance is valid!



# "\$schema" keyword

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  ...  
}
```

The **\$schema** keyword says: This object is a JSON schema, conforming to the schema at <http://json-schema.org/draft-04/schema#>.

# "type" keyword

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "type",  
  ...  
}
```

The *type* keyword says: The JSON instance must be of this *type*. Types:

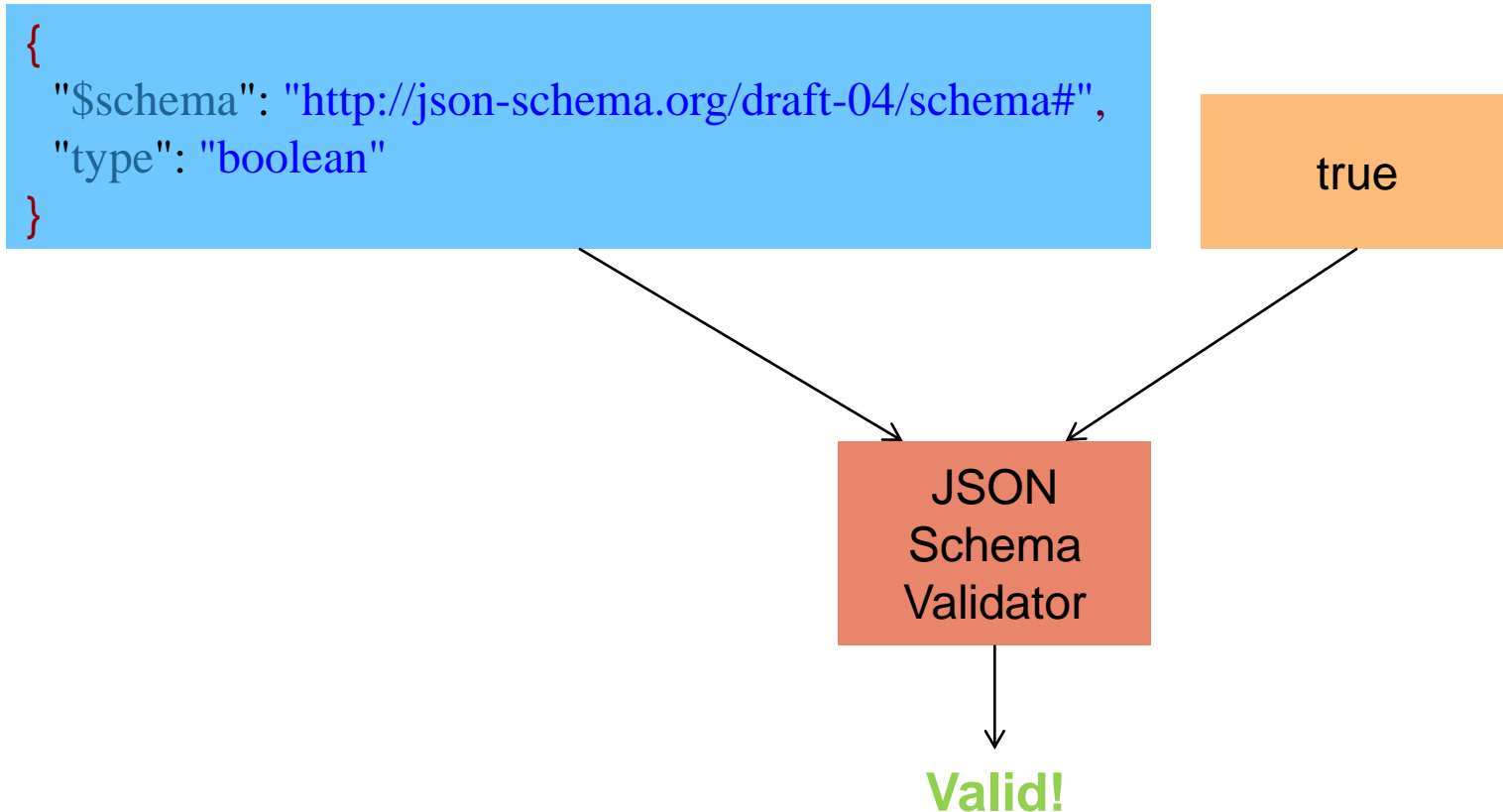
- "boolean"
- "number"
- "string"
- "array"
- "object"
- "integer"
- "null"

# Boolean type

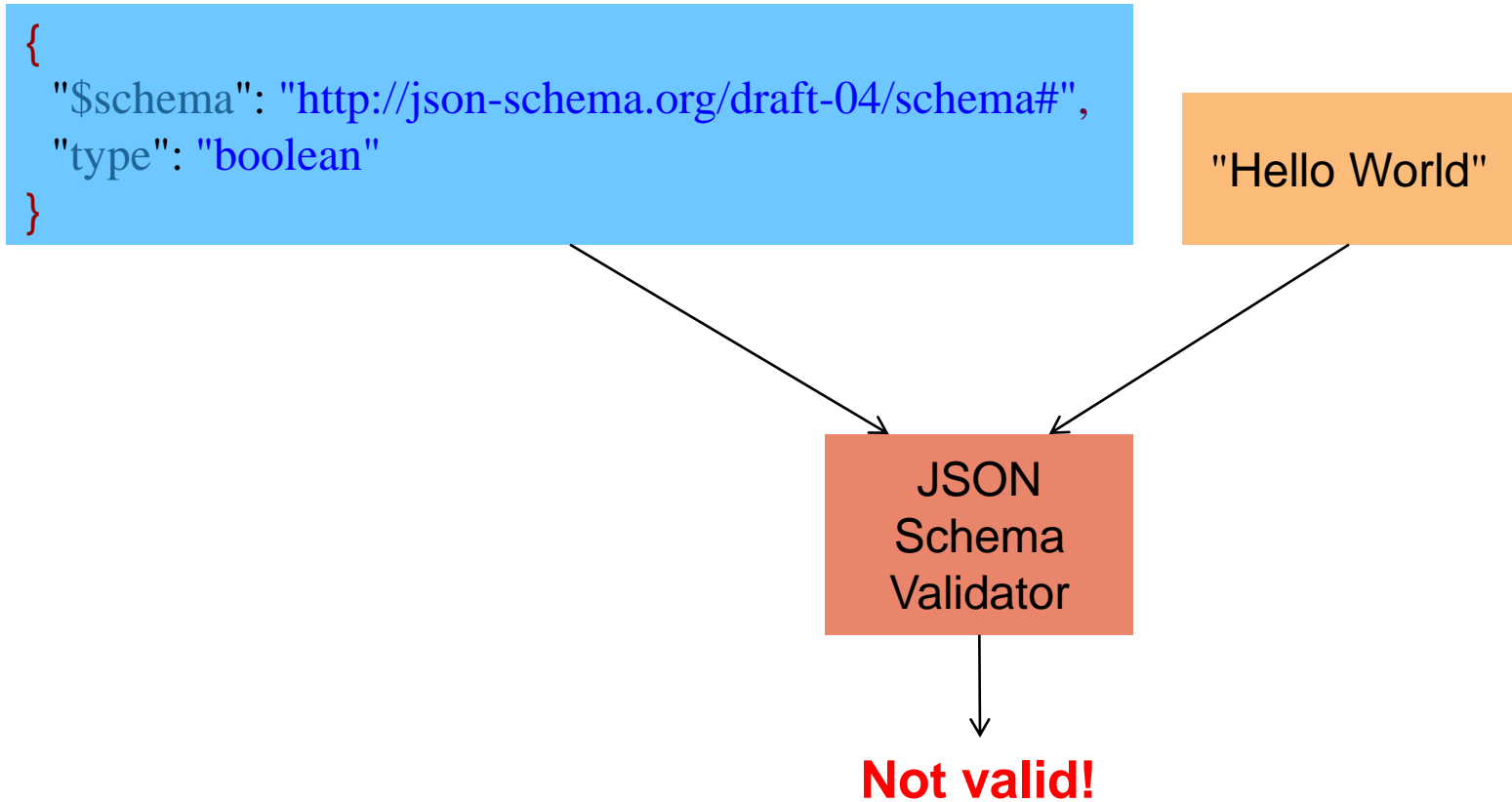
```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "boolean"  
}
```

This schema constrains instances to contain only a boolean value (true or false).

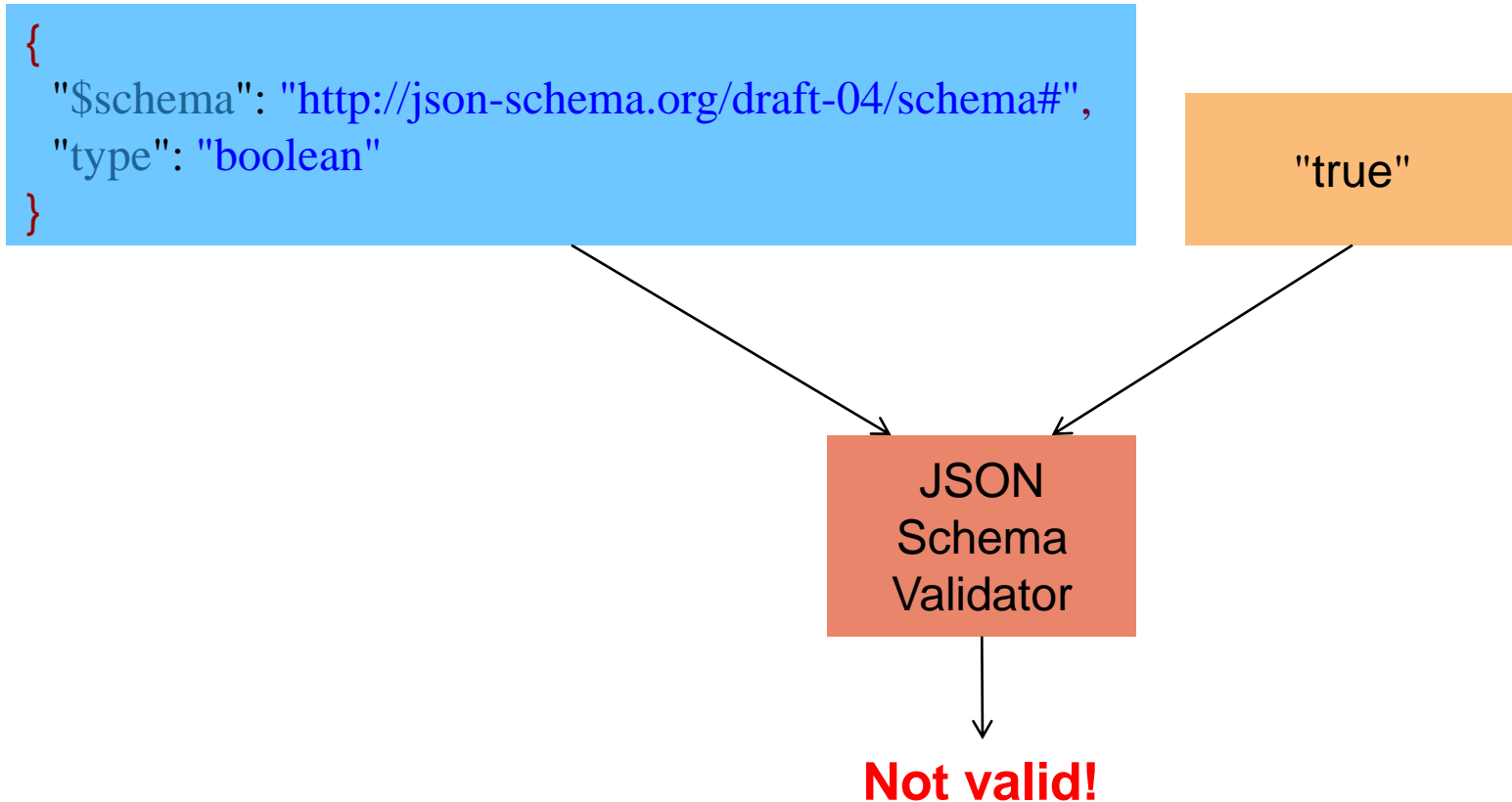
# Only boolean instances are valid



# String instances are not valid



# "true" is not a Boolean value





# <http://json-schema-validator.herokuapp.com/index.jsp>

This page allows you to validate your JSON instances. Paste your schema and data in the appropriate text areas and press the `Validate` button. Notes:

- inline dereferencing (using `id`) is disabled for security reasons;
- **Draft v4 is assumed.** If you want to use a draft v3 schema, add a `$schema` at the root of your schema, with `http://json-schema.org/draft-03/schema#` as a value.

Software used: [json-schema-validator](#).

Schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "boolean"
}
```

Paste schema here 1

Data:

```
true
```

Paste instance here 2

Press the Validate button 3

([load sample data](#))

Validation results: **success**

[ ]

Instance is valid!

4

# Order of keywords is irrelevant

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "boolean"  
}
```



```
{  
  "type": "boolean",  
  "$schema": "http://json-schema.org/draft-04/schema#"  
}
```

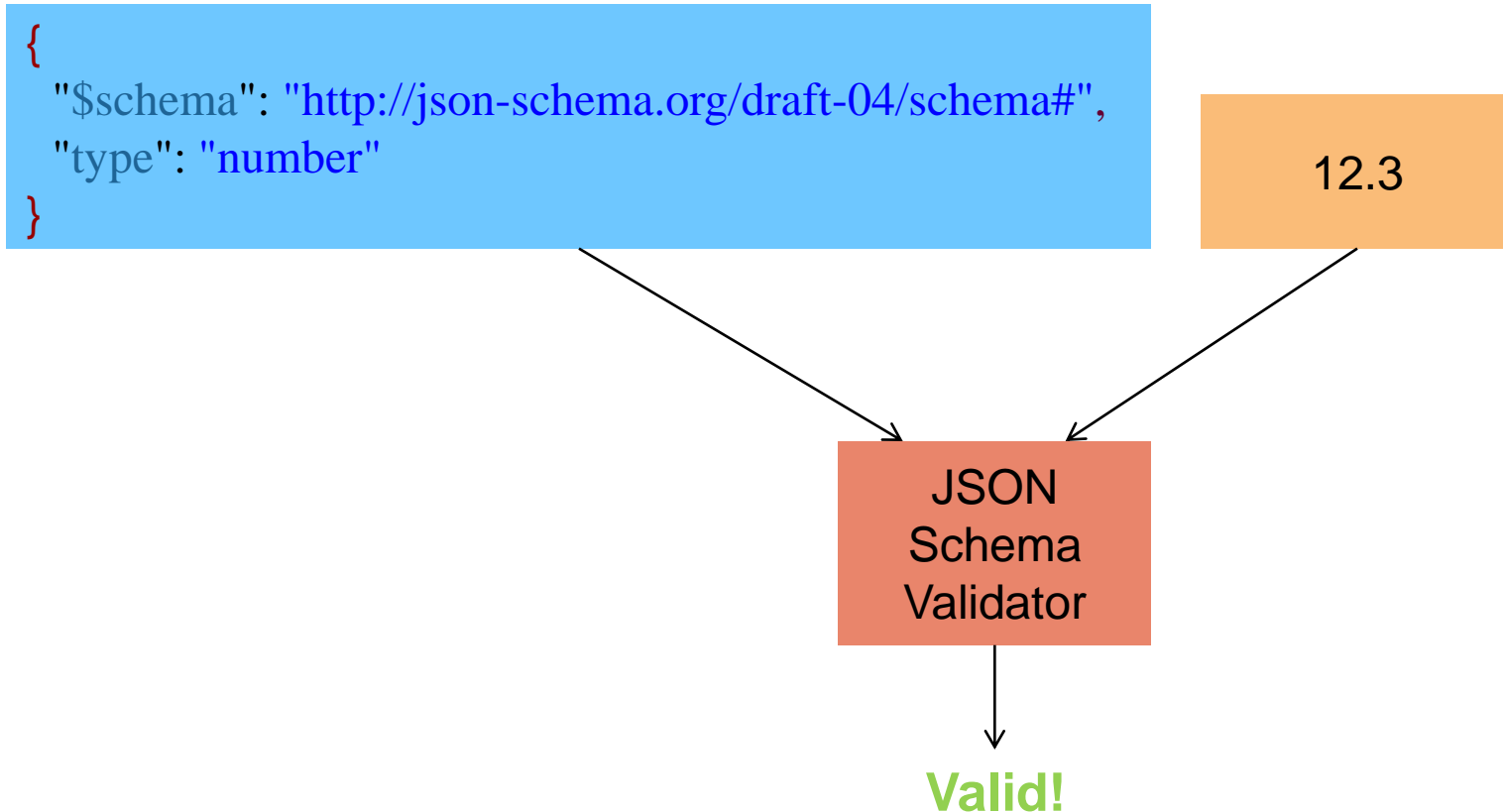
# Number type

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number"  
}
```

This schema constrains instances to contain only a number (integer, decimal, number with exponent).



# Only numeric instances are valid



# Examples of numeric values

12    99.1     $\underbrace{12.123e3}_{12.123 \times 10^3}$     12.123E3

# "enum" keyword

- The value of enum is an array.
- The items in the array is a list of values that instances may have.
- The following schema says that the only allowable values in instances are the numbers: 2, 4, 6, 8, 10.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number",  
  "enum": [2, 4, 6, 8, 10]  
}
```

# "minimum" and "maximum" keywords

- A range of values can be specified using the "minimum" and "maximum" keywords.
- The following schema constrains instances to numbers in the range 0-100, inclusive.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number",  
  "minimum": 0,  
  "maximum": 100  
}
```



# "exclusiveMinimum", "exclusiveMaximum"

- A range's endpoints can be made exclusive by using the "exclusiveMinimum" and "exclusiveMaximum" keywords.
- The following schema constrains instances to numbers in the range 0-100, exclusive.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number",  
  "minimum": 0,  
  "maximum": 100,  
  "exclusiveMinimum": true,  
  "exclusiveMaximum": true  
}
```

# "multipleOf" keyword

- Instances can be constrained to a number that is a multiple of a number.
- The following schema says that a JSON instance must be a number 0-100, inclusive, and must be a multiple of 2.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number",  
  "minimum": 0,  
  "maximum": 100,  
  "multipleOf": 2  
}
```

Here are four schema-valid values: 0    2    4    100

# **"multipleOf" value**

**The value of "multipleOf" must be greater than 0 (no negative numbers).**

# Integer type

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "integer"  
}
```

This schema constrains instances to contain only an integer.

Here are two schema-valid values: -900    129

# Number constraints also apply to integer

The constraints that apply to "number" also apply to "integer":

- enum
- minimum
- maximum
- exclusiveMinimum
- exclusiveMaximum
- multipleOf

# String type

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string"  
}
```

This schema constrains instances to contain only a string.

# "maxLength" keyword

The maximum length of a string is constrained using the "maxLength" keyword.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "maxLength": 20  
}
```

Here is a schema-valid value: "Hello World"

# Value of "maxLength"

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "maxLength": _____  
}
```

↑  
Must be an integer,  $\geq 0$



# "minLength" keyword

- The "minLength" keyword is used to specify the shortest string length allowed.
- The value of "minLength" must be an integer, greater than or equal to 0.
- **The default value is 0.**

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "minLength": 5,  
  "maxLength": 20  
}
```

# "pattern" keyword

- The set of characters that can be used in a string can be constrained using the "**pattern**" keyword, whose value is a regular expression.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "maxLength": 20,  
  "pattern": "^[a-zA-Z]*$"  
}
```

# "pattern" value is a regular expression

- The value of "pattern" is a regular expression.
- The regular expressions are not implicitly anchored so you must use the start and end anchors (^...\$).

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "maxLength": 20,  
  "pattern": "^[a-zA-Z ]*$"  
}
```

This symbol indicates that a string must *end* with the letters of the alphabet plus space.

This symbol indicates that a string must *start* with the letters of the alphabet plus space.

# How to read this JSON Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "maxLength": 20,  
  "pattern": "^[a-z A-Z ]*$"  
}
```

“The string in a JSON instance **cannot have a length greater than 20 characters** and the characters must be a-z, A-Z, or space.

”



# Components of regular expressions

Basic pattern	Matching string
$x$	The character $x$
$\cdot$	Any character, except newline
$[xyz \dots]$	Any of the characters, $x, y, z, \dots$
Repetition operators:	
$R?$	$R$ is optional (zero or one occurrence)
$R^*$	Zero or more occurrences of $R$
$R^+$	One or more occurrences of $R$
Compositional operators:	
$R_1R_2$	An $R_1$ followed by an $R_2$
$R_1 R_2$	Either an $R_1$ or an $R_2$
Grouping:	
$(R)$	$R$ itself

## Legend:

$x, y, z, \dots$  stand for any character

$R, R_1, R_2, \dots$  stand for any regular expression

## Operator Precedence:

Repetition operators have the highest precedence (bind most tightly); next comes the concatenation operator; and the alternatives operator  $|$  has the lowest precedence

# Enumerate the allowed string

- The "enum" keyword can be used with the string type.
- The following schema says that only three strings are allowed: "red", "white", or "blue".

```
{  
"$schema": "http://json-schema.org/draft-04/schema#",  
"type": "string",  
"enum": ["red", "white", "blue"]  
}
```

Here's a schema-valid instance: "red"

# Escaping the dot (.) and the dash (-)

- In regular expressions the dot (.) symbol means “**any character**”
- Suppose that you want the dot (.) symbol, not “any character”. Here’s how to do it: `\\.`
- Similarly, in regular expressions the dash (-) symbol means “**range-of-characters**”
- Suppose that you want the dash (-) symbol, not “**range-of-characters**”. Here’s how to express it: `\\-`



# **"enum" applies to all types**

**The "enum" keyword can be used with all seven types.**

# "enum" without "type"

- When the "type" keyword is specified, then all the values in the "enum" array must conform to that type.
- If the "type" keyword is **omitted**, then the values in the "enum" array can be of any type.
- The following schema says that a JSON instance can be the string: "red", the number 12, or the boolean false.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "enum": ["red", 12, false]  
}
```

# **date-time format**

**A string with format date-time must conform to one of these two forms:**

`yyyy-MM-dd'T'HH:mm:ssZ`

`yyyy-MM-dd'T'HH:mm:ss.SSSZ`

# "format" keyword

- The "format" keyword is used to specify the format of a string.
- The following schema says that JSON instances must consist of a string and the string must be formatted as an ISO 8601 date/time value.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "string",  
  "format": "date-time"  
}
```

Here's a schema-valid instance: **"2014-06-20T12:50:00Z"**

# "format" keyword values

- Valid values for the "format" keyword:
  - "date-time"
  - "email"
  - "hostname"
  - "ipv4"
  - "ipv6"
  - "uri"
- Those values are called *format attributes*.
- The "format" keyword may only be used with the string type.

# Examples of "format" values

- "date-time"  
"2014-06-20T12:50:00Z"
- "email"  
"smith@example.org"
- "hostname"  
"www.google.com"
- "ipv4"  
"192.168.5.0"
- "ipv6"  
"2001:db8:a0b:12f0::1"
- "uri"  
"http://www.google.com"

# Array type

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array"  
}
```

This schema constrains instances to contain only an array.

Here's a schema-valid value:

```
["value1", "value2", 12, null]
```

# Enumerate the allowed arrays

- The "enum" keyword can be used with the array type.
- The following schema says that instances may be either of these arrays: ["A", "B"] or [1,2,3].

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "enum": [["A", "B"], [1,2,3]]  
}
```

Here are the only two valid instance values: ["A", "B"] [1,2,3]



# "maxItems" keyword

- The maximum number of items in the array is specified using the "maxItems" keyword.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "maxItems": 3  
}
```

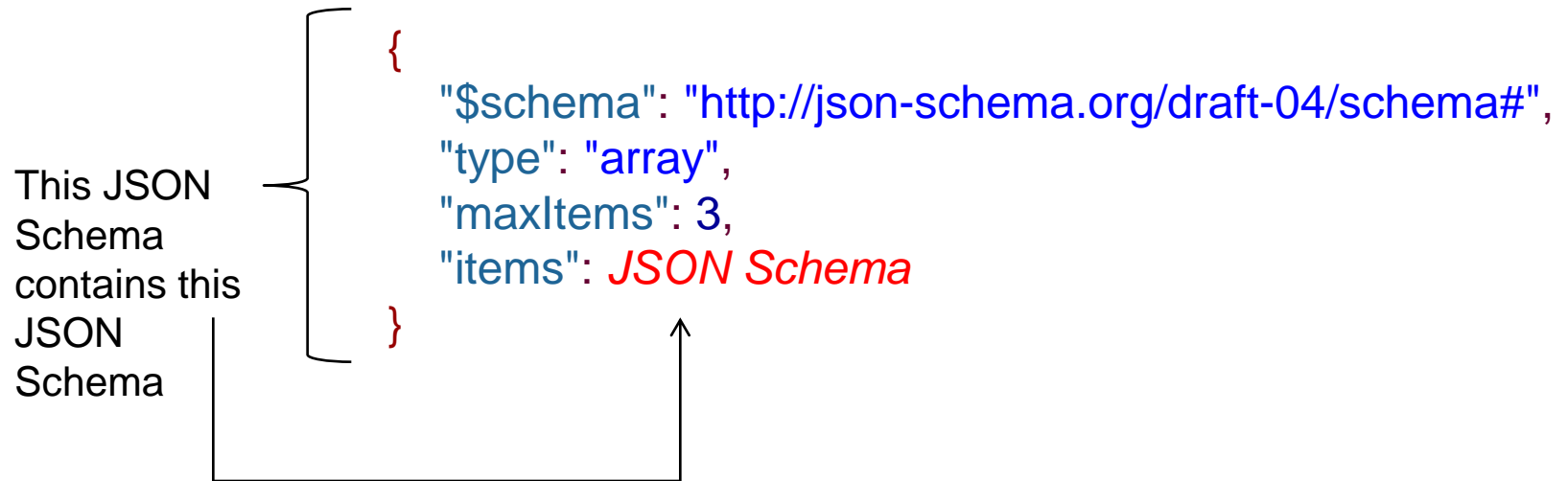
Here's a schema-valid instance: [ 1, true, ["A", "B"] ]

# "items" keyword

- The items in an array can be constrained using the "items" keyword.
- The value of "items" is a JSON Schema.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "maxItems": 3,  
  "items": JSON Schema  
}
```

# Recursive definition



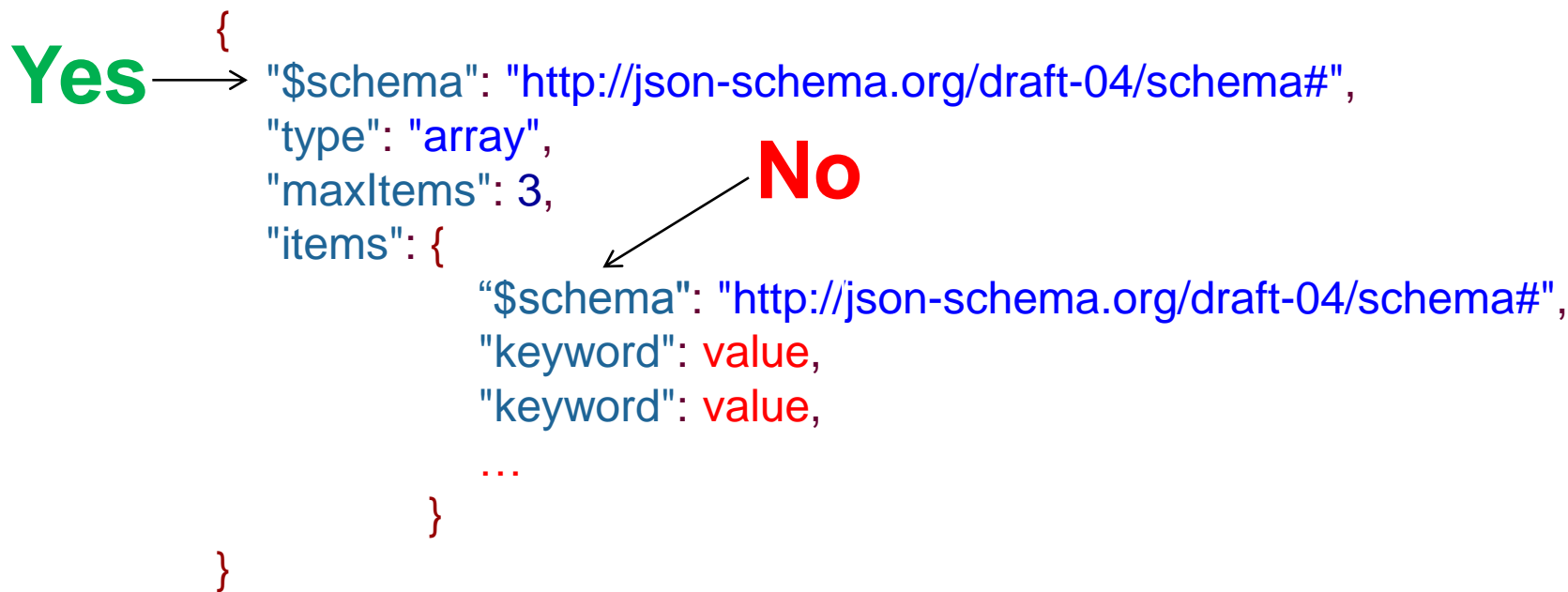
# subschema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "maxItems": 3,  
  "items": {  
    "keyword": value,  
    "keyword": value,  
    "keyword": value,  
    ...  
  }  
}
```

↖  
root schema

↑  
subschema

# Don't use \$schema in subschemas



"\$schema" should only be used in the root schema, not in subschemas. It's **not illegal in draft #4**.

But in draft v5, **\$schema appearing in a subschema will be an error**.

# Default value for "items"

If "items" is not present, it has a default value of { }, which means that each item in the array may be any value.

# Max of 3 items, each an integer

The following schema says that instances must not contain more than 3 items and each item must be an integer.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "maxItems": 3,  
  "items": {  
    "type": "integer"  
  }  
}
```

Four schema-valid instances: [1,2,3]    [1,2]    [1]    []

# Max of 3 items, each an integer 0 – 100

The following schema says that instances must not **contain more than 3 items** and each item must be an integer and the integer must be between 0 and 100.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "array",
  "maxItems": 3,
  "items": {
    "type": "integer",
    "minimum": 0,
    "maximum": 100
  }
}
```



# "uniqueItems" keyword

**"uniqueItems"** keyword. → each item in the array be unique.

The **default** value is **false**.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "maxItems": 3,  
  "items": {  
    "type": "integer",  
    "minimum": 0,  
    "maximum": 100  
  },  
  "uniqueItems": true  
}
```

**Max of 3 items, each a string with max length 20, consisting of letters and spaces only**

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "maxItems": 3,  
  "items": {  
    "type": "string",  
    "maxLength": 20,  
    "pattern": "^[a-zA-Z ]*$"  
  }  
}
```

# Legal values for "items"

- The value of "items" is **an object (a JSON Schema)**.
- Alternatively, the value of "items" **may be an array**. Each item in the array must be an object (a JSON Schema).

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "items": JSON Schema  
}
```

This kind of array is  
called an *array tuple*

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "items": [  
    JSON Schema,  
    JSON Schema,  
    ...  
  ]  
}
```

# Array may contain two strings

- The below schema specifies an array consisting of up to two items.
- If the first item is present, it must be either the string "white" or "black".
- If the second item is present, it must be either the string "cup" or "plate".
- The array may be empty.


```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "array",
  "items": [
    { "type": "string", "enum": ["white", "black"] },
    { "type": "string", "enum": ["cup", "plate"] }
  ]
}
```

# Array has open content

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "items": [  
    { "type": "string", "enum": ["white", "black"] },  
    { "type": "string", "enum": ["cup", "plate"] }  
  ]  
}
```

This schema specifies what the first two items in the array must be, but it doesn't say there can't be additional items.

This is a valid value: [ "white", "cup", true, null, {"foo": 12} ]

  
additional items

# "additionalItems" keyword

- **"additionalItems": false** is used to specify that an array cannot contain additional items.
- Now the array is constrained to no more than two items:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "items": [  
    { "type": "string", "enum": ["white", "black"] },  
    { "type": "string", "enum": ["cup", "plate"] }  
  ],  
  "additionalItems": false  
}
```

# Array tuple with { } for each value

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "items": [  
    {},  
    {}  
  ],  
  "additionalItems": false  
}
```

The array must contain up to two items.

**Each item can be anything: a number, integer, boolean, array, object, or null.**

Valid instances: [ ] [1] [1, true] [1, [1, true]]





# additionalItems with a JSON Schema value

- We have seen that `additionalItems` can have a boolean value.
- Alternatively, it can have a value that is an object (JSON Schema)
- `"additionalItems": JSON Schema`

# Any number of boolean values

- The below schema specifies that an instance must be an array that contains one item: a string of max length 20, consisting of the lower and uppercase letters plus space.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "array",  
  "items": [  
    { "type": "string", "maxLength": 20, "pattern": "^[a-zA-Z ]*$" }  
  ],  
  "additionalItems": { "type": "boolean" }  
}
```

valid instances:

```
["Hello World", true, false, false, true]  
["Hello World"]  
[]
```

# Object type

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object"  
}
```

The schema constrains instances to contain only an object.

Here's a schema-valid value:

```
{  
  "X": 12,  
  "Y": "hello"  
}
```

# Enumerate the allowed objects

- The "enum" keyword can be used to enumerate the objects that are allowed in instances.
- The following schema enumerates two objects:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "enum": [  
    {"name": "John Doe", "age": 30},  
    {"company": "Google", "product": "searching"}  
  ]  
}
```

Here are the only valid instances:

```
{"name": "John Doe", "age": 30}    {"company": "Google", "product": "searching"}
```

# Specify the property names, but values are variable

- The "enum" keyword specifies both property name and property value.
- enumerate the property names but leave the property values variable.

# "properties" keyword

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": {"type": "string"},  
    "age": {"type": "integer"}  
  }  
}
```

Instances must be an object. The object may contain a property "name" and a property "age".

The value of "name" must be a string. The value of "age" must be an integer.

# Sample instance

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": {"type": "string"},  
    "age": {"type": "integer"}  
  }  
}
```

Here is a valid instance:

```
{  
  "name": "John Doe",  
  "age": 30  
}
```

# Property name can be an empty string

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": {"type": "string"},  
    "age": {"type": "integer"},  
    "": {"type": "boolean"}  
  }  
}
```

Here is a valid instance:

```
{  
  "name": "John Doe",  
  "age": 30,  
  "": true  
}
```



# Mandate properties using "required"

- The **"required"** keyword is used to mandate properties.
- The value of **"required"** is an array of strings, each string corresponding to the name of a property.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": { "type": "string" },  
    "age": { "type": "number" }  
  },  
  "required": ["name", "age"]  
}
```

# Object has open content by default

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": { "type": "string" },  
    "age": { "type": "number" }  
  },  
  "required": ["name", "age"]  
}
```

The schema specifies two properties that must be in instances. But it doesn't say there can't be additional properties. This is a valid value:

```
{  
  "name": "John Doe",  
  "age": 30,  
  "height": 68,  
  "married": true  
}
```

} additional properties

# "additionalProperties" keyword

Instances can be constrained to contain only those listed in the schema by using **"additionalProperties": false**

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": { "type": "string" },  
    "age": { "type": "number" }  
  },  
  "required": ["name", "age"],  
  "additionalProperties": false  
}
```

Now instances must contain "name" and "age" and nothing else.

# Well-constrained object

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "name": { "type": "string", "maxLength": 20, "pattern": "[a-zA-Z ]*$" },  
    "age": { "type": "number", "minimum": 0, "maximum": 125 }  
  },  
  "required": ["name", "age"],  
  "additionalProperties": false  
}
```

Instances must contain an object.

The object must have exactly two properties: one named "name" and the other named "age".

The value of "name" must be a string, not longer than 20 characters, consisting of the symbols a-z, A-Z, and space.

The value of "age" must be a number in the range 0 – 125, inclusive.

# Must specify "type": "object" with "properties"

When "properties" is used, then "type": "object" is required to be present. If it is not present:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "properties": {  
    "name": {"type": "string"},  
    "age": {"type": "integer"}  
  }  
}
```

then any value in the instance is valid.

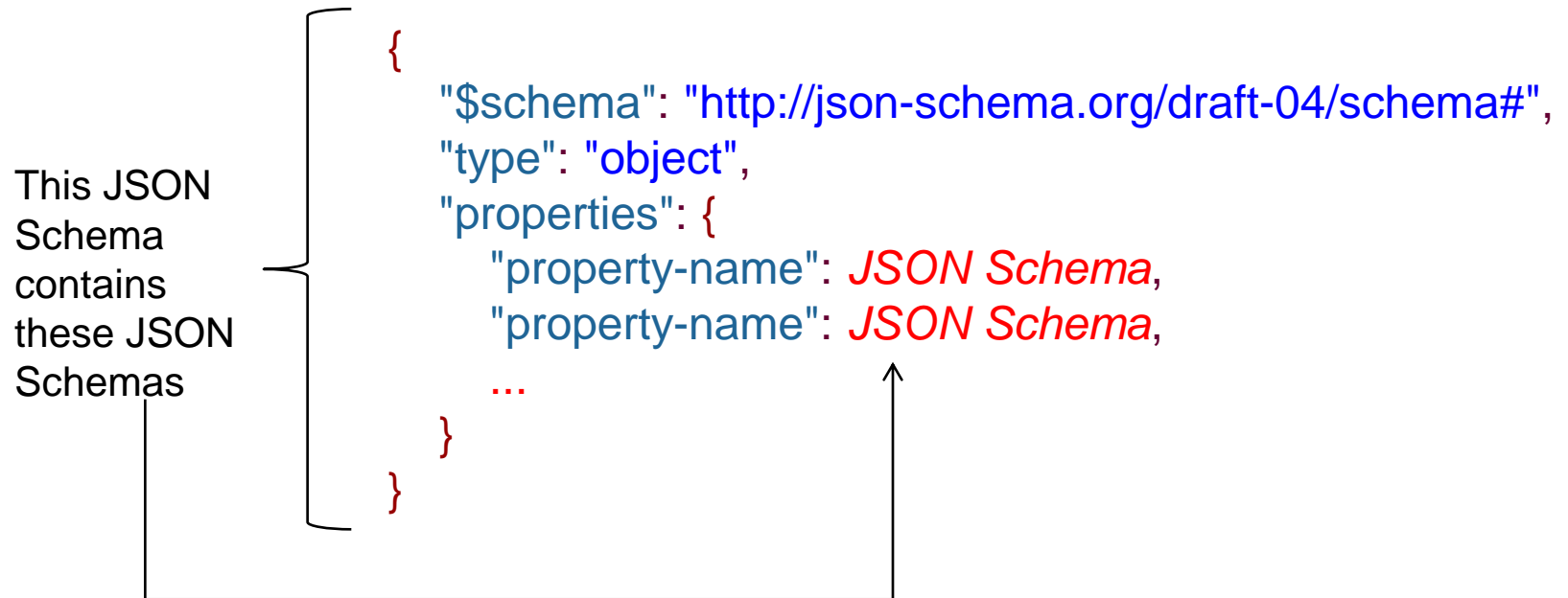


# Summary of "properties"

The value of "properties" is an object. The object contains members. Each member has a name and its value is a schema:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "property-name": JSON Schema,  
    "property-name": JSON Schema,  
    ...  
  }  
}
```

# Recursive definition!





# additionalItems versus additionalProperties

Distinguish between "additionalItems": false and "additionalProperties": false:

- The former is used to disallow additional items in an array, the latter is used to disallow additional properties in an object

# Constraining the number of properties

- The minimum and maximum number of properties can be constrained using the "**minProperties**" and "**maxProperties**" keywords.
- The following schema specifies a list of activities: **bowling, golfing, and kayaking**. At least one activity but not more than two must be selected for inclusion in a JSON instance:

# Sample instance

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "bowling": { "type": "string", "maxLength": 20 },  
    "golfing": { "type": "string", "maxLength": 20 },  
    "kayaking": { "type": "string", "maxLength": 20 }  
  },  
  "minProperties": 1,  
  "maxProperties": 2,  
  "additionalProperties": false  
}
```

This JSON instance is valid since it has two of the activities:

```
{  
  "bowling": "50 Lanes",  
  "kayaking": "Red Rock River"  
}
```

# Valid instances

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "age": {"type": "integer"}
  },
  "required": ["name", "age"],
  "additionalProperties": {
    "type": "boolean"
  },
  "maxProperties": 4
}
```

```
{
  "name": "John Doe",
  "age": 30
}
```

3

Valid instances:

```
{
  "name": "John Doe",
  "age": 30,
  "married": true,
  "living": true
}
```

1

```
{
  "name": "John Doe",
  "age": 30,
  "married": true,
}
```

2

# Regex for the property name

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "age": {"type": "integer"}
  },
  "required": ["name", "age"],
  "patternProperties": {
    "^(married|divorced|living)$": {"type": "boolean"}
  },
  "additionalProperties": false
}
```

# Valid, invalid instance

## Valid instance:

```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "living": true  
}
```



# Instance may be a boolean or a number

- The value of "type" can be an array of types.
- This schema says that JSON instances must contain either a boolean value or a number:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": [ "boolean", "number" ]  
}
```

Here are two valid values: **true**      **12**



# Instance may be an array or an object

- This says that JSON instances must be either an array or an object:

`"type": ["array", "object"]`

- The following schema constrains the array, if present, to 3 integers in the range 0-100, and constrains the object, if present, to exactly two properties – name and age – and no others:

# Well-constrained array or object

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": ["array", "object"],
  "maxItems": 3,
  "items": {
    "type": "integer",
    "minimum": 0,
    "maximum": 100
  },
  "properties": {
    "name": { "type": "string", "maxLength": 20, "pattern": "[a-zA-Z ]+" },
    "age": { "type": "integer", "minimum": 0, "maximum": 100 }
  },
  "required": ["name", "age"],
  "additionalProperties": false
}
```

Here are two valid values: [12, 3, 99]

```
{
  "name": "John Doe",
  "age": 30
}
```



# Comments for humans

The "**description**" keyword is used to provide a human-readable description.

The value of "**description**" is any string. It is ignored in validation.

The "**title**" keyword is a short human-readable title.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "Short title",  
  "description": "This description can be arbitrarily long "  
}
```

# not

- The "not" keyword is used to specify what you don't want.
- This schema says that JSON instances can contain any value, as long as it's *not* a string:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "not": { "type": "string" }  
}
```

Here are two valid values: `true`      `["Hello", "World"]`

# Creating a repository of schemas

The **"definitions"** keyword is used to define a repository of schemas.

The value of **"definitions"** is a JSON object which contains one or more properties and the value of each property is a schema:

```
"definitions": {  
    "property1": JSON Schema  
    "property2": JSON Schema  
    ...  
}
```

# Repository with 2 schemas

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "person":
    {
      "type": "object",
      "properties": {
        "name": { "type": "string", "maxLength": 20, "pattern": "[a-zA-Z]*" },
        "age": { "type": "number", "minimum": 0, "maximum": 120 }
      },
      "required": [ "name", "age" ],
      "additionalProperties": false
    },
    "evens":
    {
      "type": "number", "multipleOf": 2
    }
  }
}
```

← 2 schemas





# Referencing a schema

- The "\$ref" keyword is used to reference a schema:
- "\$ref": *reference to a json-schema*
- The value of "\$ref" is a path expression (very similar to path expressions in XPath).
- A json-schema validator replaces "\$ref" and its value by the schema that it references.

# Internal reference

- The "\$ref" in the following schema references the evens schema.
- The # symbol indicates an internal reference; specifically, it references the schema within "evens" that is within "definitions":

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "person": {
      "type": "object",
      "properties": {
        "name": { "type": "string", "maxLength": 20, "pattern": "[a-zA-Z ]*$" },
        "age": { "type": "number", "minimum": 0, "maximum": 120 }
      },
      "required": [ "name", "age" ],
      "additionalProperties": false
    },
    "evens": {
      "type": "number", "multipleOf": 2
    }
  },
  "$ref": "#/definitions/evens"
}
```

# External reference

**"\$ref"** can reference a schema in another document, as shown here:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "$ref": "definitions.json#/definitions/evens"  
}
```

# Here's how to reference the values

```
{  
  "foo": ["bar", "baz"],  
  "": 0,  
  " ": 7,  
}
```

Path	What is referenced
#	the whole document
#/foo	["bar", "baz"]
#/foo/0	"bar"
#/	0
#/%20	7

# Choice of schemas

A choice of schemas can be created using the "oneOf" keyword. The value of "oneOf" is an array and each item in the array must be a schema:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "oneOf": [  
    JSON Schema,  
    JSON Schema,  
    ...  
  ]  
}
```

# Either a number or a boolean

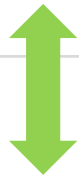
**This schema says that a JSON instance must contain either a number or a boolean:**

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "oneOf": [  
    {"type": "number"},  
    {"type": "boolean"}  
  ]  
}
```

Here are two valid values:    **12**    **true**

# Equivalent

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "oneOf": [  
    {"type": "number"},  
    {"type": "boolean"}  
  ]  
}
```



***equivalent***

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": [ "boolean", "number" ]  
}
```

# A multiple of 3 or multiple of 5

**This schema says that a JSON instance must contain a number that is either a multiple of 3 or a multiple of 5:**

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "number",  
  "oneOf": [  
    { "multipleOf": 3 },  
    { "multipleOf": 5 }  
  ]  
}
```

Here are two valid values:

9

10



# Value of the "allOf" keyword

- The value of the **"allOf" keyword is an array.**
- Each item in the array is a schema.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "allOf": [  
    JSON Schema,  
    JSON Schema,  
    ...  
  ]  
}
```

# Valid instance

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "person": {
      "type": "object",
      "properties": {
        "name": { "type": "string", "maxLength": 20, "pattern": "^([a-zA-Z])*$" },
        "age": { "type": "number", "minimum": 0, "maximum": 120 }
      },
      "required": [ "name", "age" ]
    },
    "evens": { "type": "number", "multipleOf": 2 }
  },
  "allOf": [
    { "$ref": "#/definitions/person" },
    {
      "type": "object",
      "properties": {
        "height": { "type": "number", "minimum": 0 }
      },
      "required": [ "height" ]
    }
  ],
  "maxProperties": 3
}
```

```
{
  "name": "John Doe",
  "age": 30,
  "height": 68
}
```

JSON Schema  
Validator

valid!

# The "default" keyword

- The "default" keyword is used to provide a default value.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "title": { "type": "string", "enum": ["Mr", "Mrs", "Miss"], "default": "Mr"},
    "name": { "type": "string", "maxLength": 20, "pattern": "^[a-zA-Z ]*$" },
    "age": { "type": "number", "minimum": 0, "maximum": 120 }
  },
  "required": [ "name", "age" ],
  "additionalProperties": false
}
```

# Expressing co-constraints

- The "dependencies" keyword enables constraints between properties (a.k.a. co-constraints) to be expressed.
- Consider properties of these shapes: circle and rectangle. Data for radius and diameter must be provided for circle. Data for width and length must be provided for rectangle. So if there is a radius, there must be a diameter and vice versa. If there is a width, there must be a length and vice versa.

# Co-constraint: *“If there is a radius, then there must be a diameter and vice versa”*

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "length": { "type": "number", "minimum": 0 },  
    "width": { "type": "number", "minimum": 0 },  
    "radius": { "type": "number", "minimum": 0 },  
    "diameter": { "type": "number", "minimum": 0 }
```

```
  },
```

```
  "dependencies": {
```

```
    "radius": ["diameter"],
```

```
    "diameter": ["radius"],
```

```
    "length": ["width"],
```

```
    "width": ["length"]
```

```
  },
```

```
  "additionalProperties": false
```

```
}
```

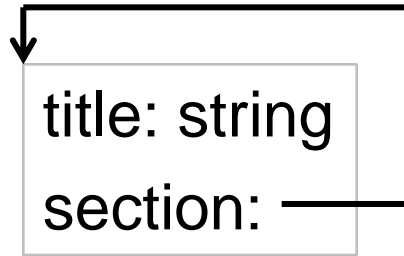
If the instance has "radius" then it must also have "diameter" and vice versa.  
If the instance has "length" then it must also have "width" and vice versa.

# Create a schema for this

```
{  
  "book": {  
    "title": "Title 1",  
    "section": {  
      "title": "Title 1.1",  
      "section": {  
        "title": "Title 1.1.1",  
        "section": {  
          "title": "Title 1.1.1.1"  
        }  
      }  
    }  
  }  
}
```

A section contains a title and a section.  
See the recursion?

# Here's what we want



# Create a recursive "definitions"

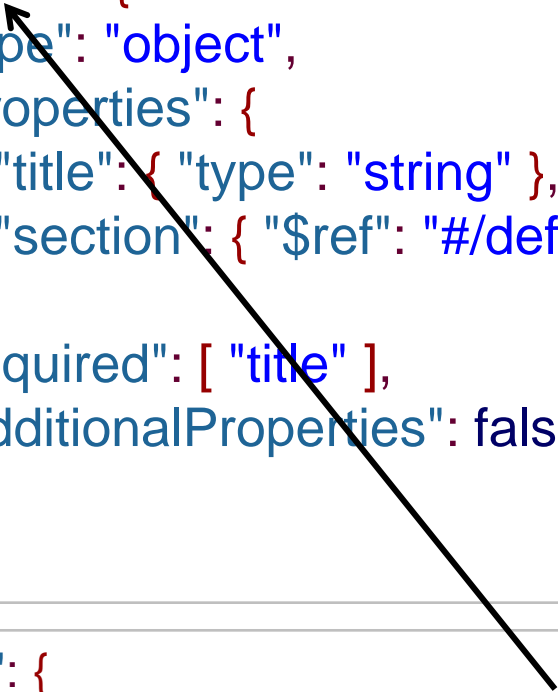
```
"definitions": {  
  "section": {  
    "type": "object",  
    "properties": {  
      "title": { "type": "string" },  
      "section": { "$ref": "#/definitions/section" }  
    },  
    "required": [ "title" ],  
    "additionalProperties": false  
  },  
},
```





# Create **"book"** and set its value to point to the recursive schema

```
"definitions": {  
  "section": {  
    "type": "object",  
    "properties": {  
      "title": { "type": "string" },  
      "section": { "$ref": "#/definitions/section" }  
    },  
    "required": [ "title" ],  
    "additionalProperties": false  
  },  
},
```



```
"properties": {  
  "book": { "$ref": "#/definitions/section" }  
}
```

# Here's the schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "section": {
      "type": "object",
      "properties": {
        "title": { "type": "string" },
        "section": { "$ref": "#/definitions/section" }
      },
      "required": [ "title" ],
      "additionalProperties": false
    }
  },
  "type": "object",
  "properties": {
    "book": { "$ref": "#/definitions/section" }
  }
}
```

# Processing JSON and JSON Schema with Java

- Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object.

<https://code.google.com/p/google-gson/>

<http://www.studytrails.com/java/json/java-google-json-introduction.jsp>

- Class JsonReader: Reads a JSON encoded value as a stream of tokens. This stream includes both literal values (strings, numbers, booleans, and nulls) as well as the begin and end delimiters of objects and arrays. The tokens are traversed in depth-first order, the same order that they appear in the JSON document. Within JSON objects, name/value pairs are represented by a single token.

<http://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/com/google/gson/stream/JsonReader.html>

# Example of XML-to-JSON

The online [freeformatter.com](https://www.freeformatter.com/xml-to-json-converter.html) tool converts this XML:

```
<Book id="MCD">  
  <Title>Modern Compiler Design</Title>  
  <Author>Dick Grune</Author>  
  <Publisher>Springer</Publisher>  
</Book>
```

to this JSON:

```
{  
  "@id": "MCD",  
  "Title": "Modern Compiler Design",  
  "Author": "Dick Grune",  
  "Publisher": "Springer"  
}
```