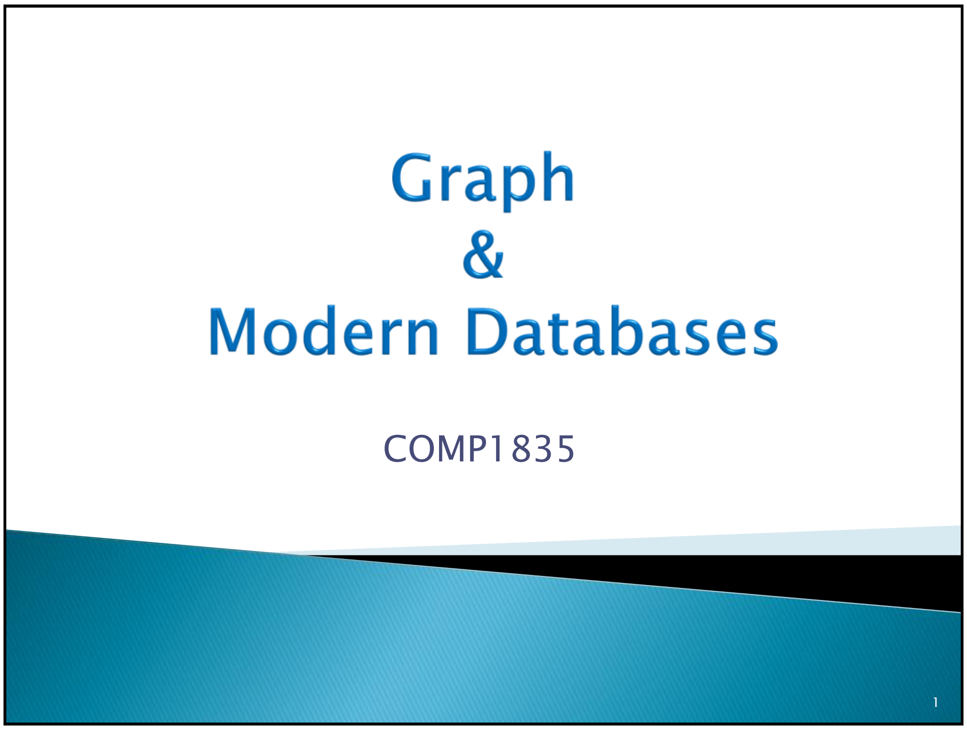
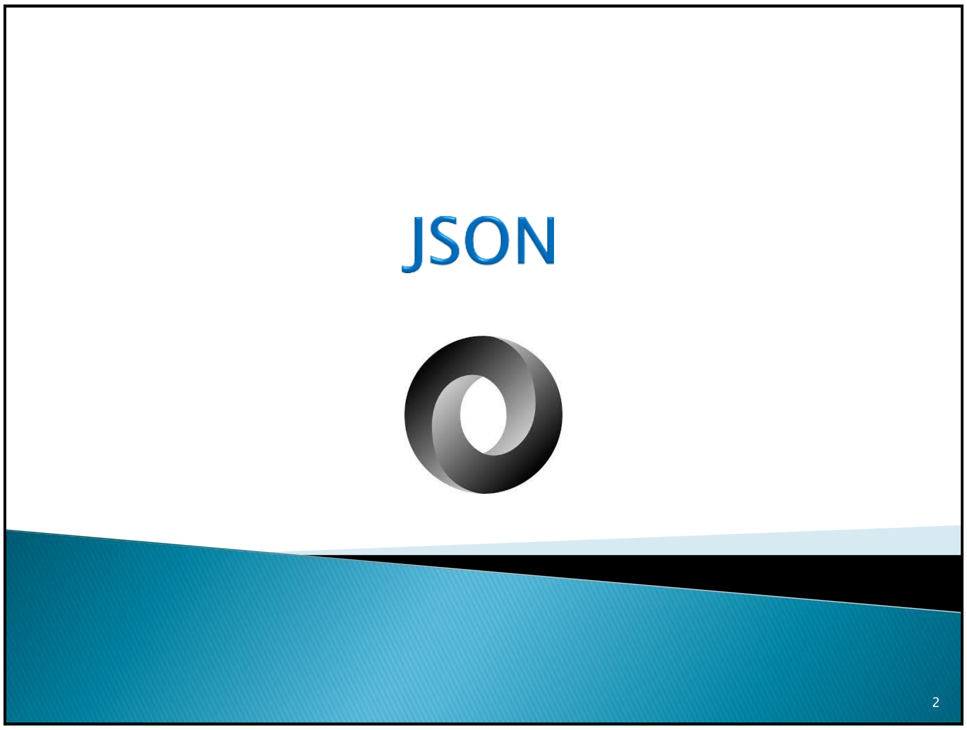


Graph & Modern Databases

COMP1835

1

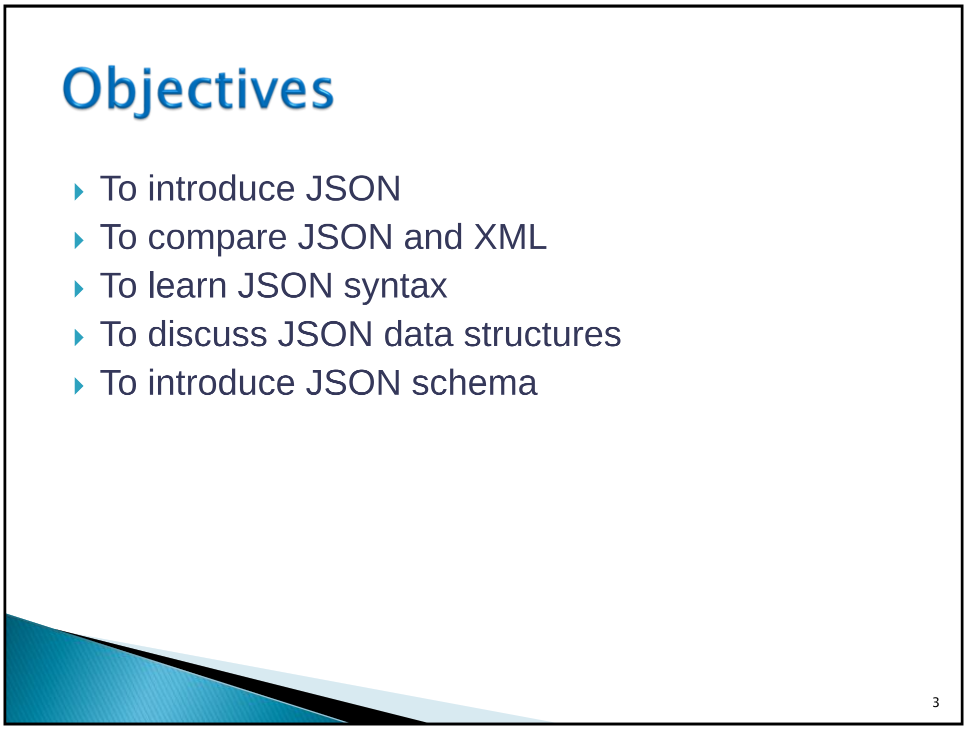
1



JSON

2

2

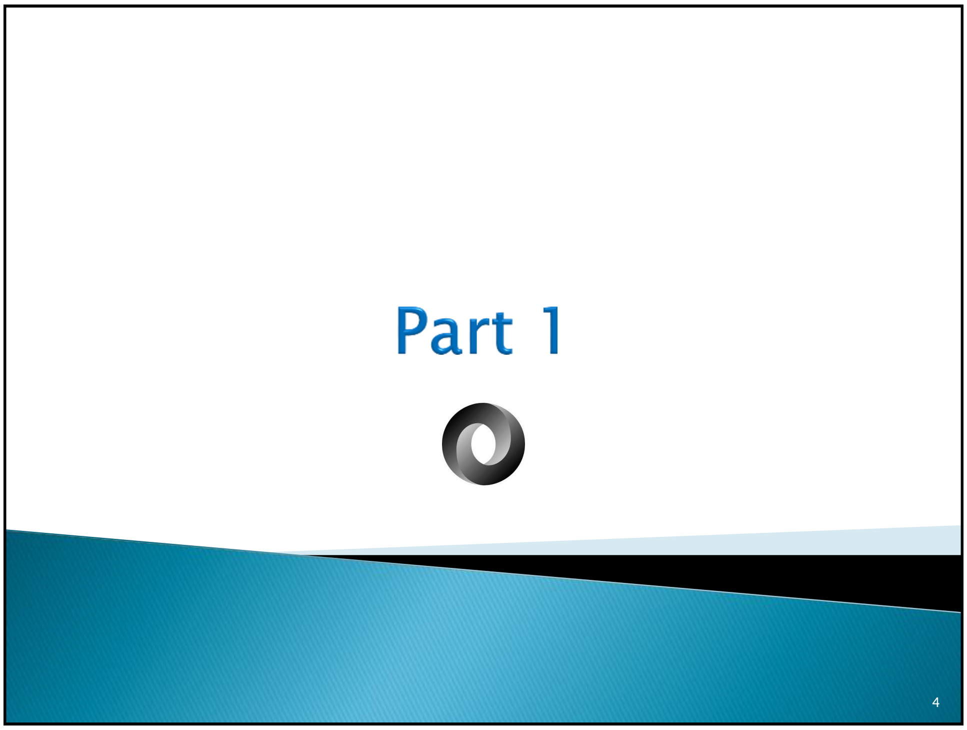


Objectives

- To introduce JSON
- To compare JSON and XML
- To learn JSON syntax
- To discuss JSON data structures
- To introduce JSON schema

3

3



Part 1

4

4

What is JSON?

- JSON
  - stands for JavaScript Object Notation
  - is a lightweight text-based open standard designed for human-readable data interchange
  - has been extended from the JavaScript scripting language.
- JSON objects are used for transferring data between server and client
- The format was specified by Douglas Crockford in early 2000s but it was first standardized in 2013
  - The latest JSON format standard was published in 2017
- The filename extension is **.json**
- The official web site: www.json.org

5

5

Where to use 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 used in many NoSQL datastores as data structure.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages, including C, C++, Java, Python, Perl, etc.

6

6

Why to use JSON

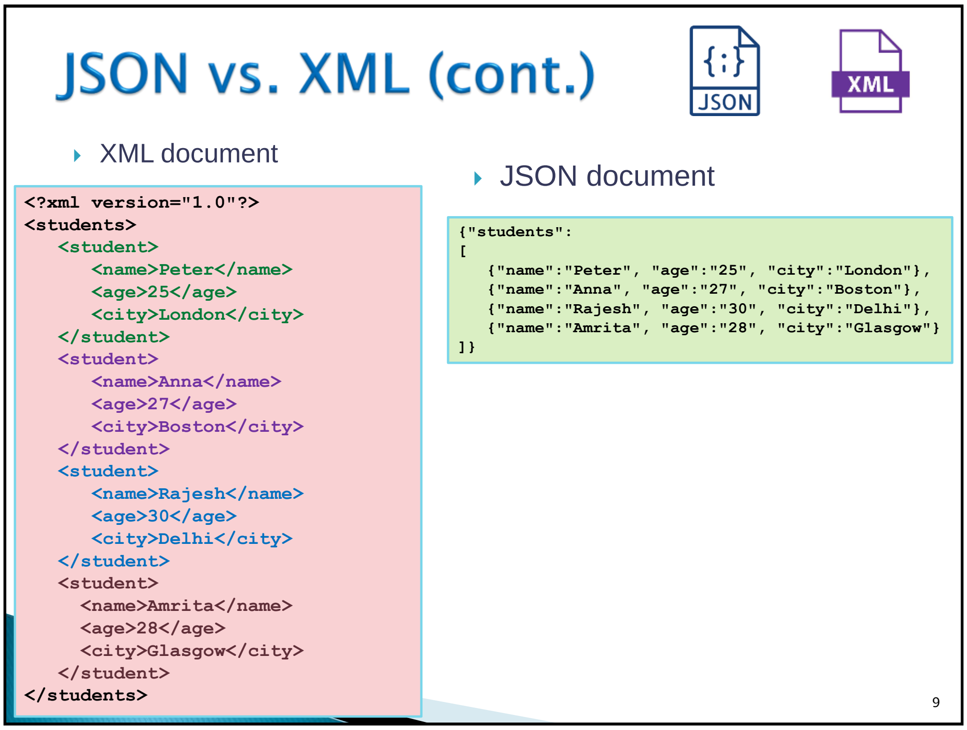
- **Standard Structure:**
  - JSON objects have standard structure that makes developers job easy to read and write code, because they know what to expect from JSON
- **Light weight**:
  - When working with NoSQL databases, it is important to load the data quickly and asynchronously. Since JSON is light weighted, it becomes easier to get and load the requested data quickly
- **Scalable**:
  - JSON is language independent, which means it can work well with most of the modern programming languages.

JSON vs. XML

- Both JSON and XML are commonly used as **data-interchange languages**
- An example: In relational database we store the records of 4 students in a table, called

`Students:`

Name	Age	City
Peter	25	London
Anna	27	Boston
Rajesh	30	Delhi
Amrita	28	Glasgow



JSON vs. XML (cont.)

- XML document

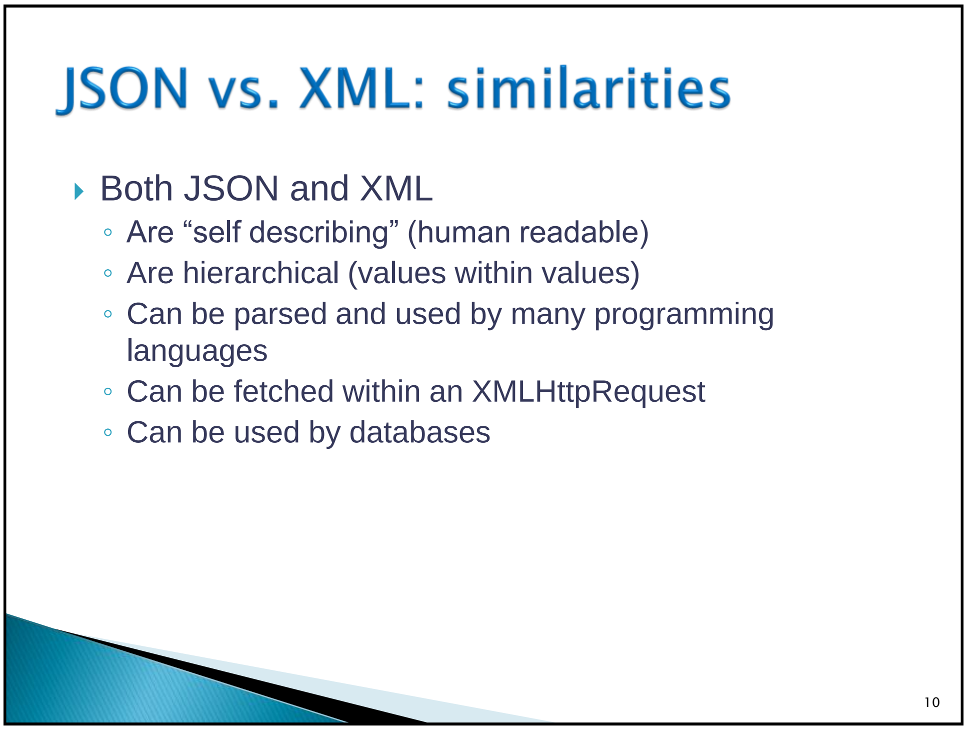
```
<?xml version="1.0"?>
<students>
   <student>
      <name>Peter</name>
      <age>25</age>
      <city>London</city>
   </student>
   <student>
      <name>Anna</name>
      <age>27</age>
      <city>Boston</city>
   </student>
   <student>
      <name>Rajesh</name>
      <age>30</age>
      <city>Delhi</city>
   </student>
   <student>
     <name>Amrita</name>
     <age>28</age>
     <city>Glasgow</city>
   </student>
</students>
```

- JSON document

```
{"students":
[
   {"name":"Peter", "age":"25", "city":"London"},
   {"name":"Anna", "age":"27", "city":"Boston"},
   {"name":"Rajesh", "age":"30", "city":"Delhi"},
   {"name":"Amrita", "age":"28", "city":"Glasgow"}
]}
```

9

9

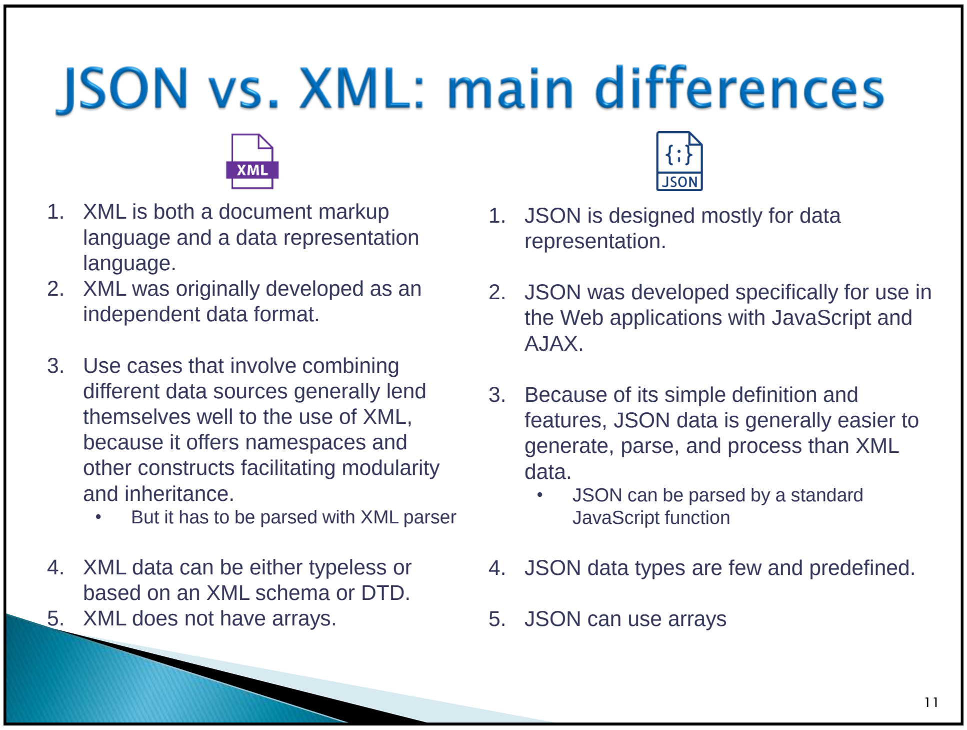


JSON vs. XML: similarities

- Both JSON and XML
  - Are “self describing” (human readable)
  - Are hierarchical (values within values)
  - Can be parsed and used by many programming languages
  - Can be fetched within an XMLHttpRequest
  - Can be used by databases

10

10



JSON vs. XML: main differences

XML

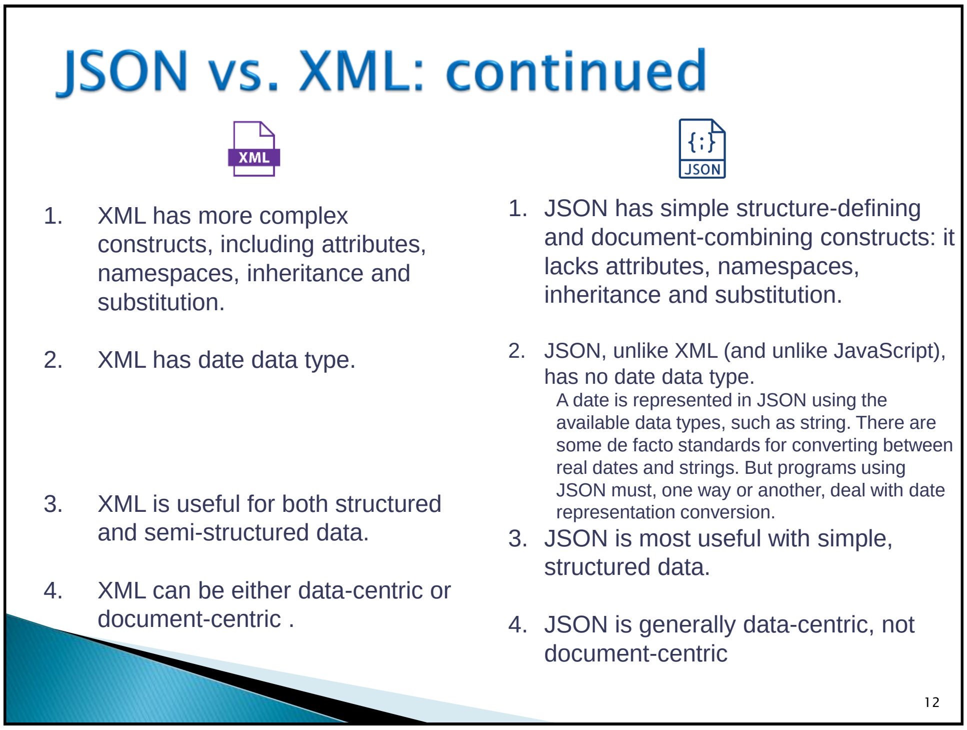
1. XML is both a document markup language and a data representation language.
2. XML was originally developed as an independent data format.
3. Use cases that involve combining different data sources generally lend themselves well to the use of XML, because it offers namespaces and other constructs facilitating modularity and inheritance.
   - But it has to be parsed with XML parser
4. XML data can be either typeless or based on an XML schema or DTD.
5. XML does not have arrays.

JSON

1. JSON is designed mostly for data representation.
2. JSON was developed specifically for use in the Web applications with JavaScript and AJAX.
3. Because of its simple definition and features, JSON data is generally easier to generate, parse, and process than XML data.
   - JSON can be parsed by a standard JavaScript function
4. JSON data types are few and predefined.
5. JSON can use arrays

11

11



JSON vs. XML: continued

XML

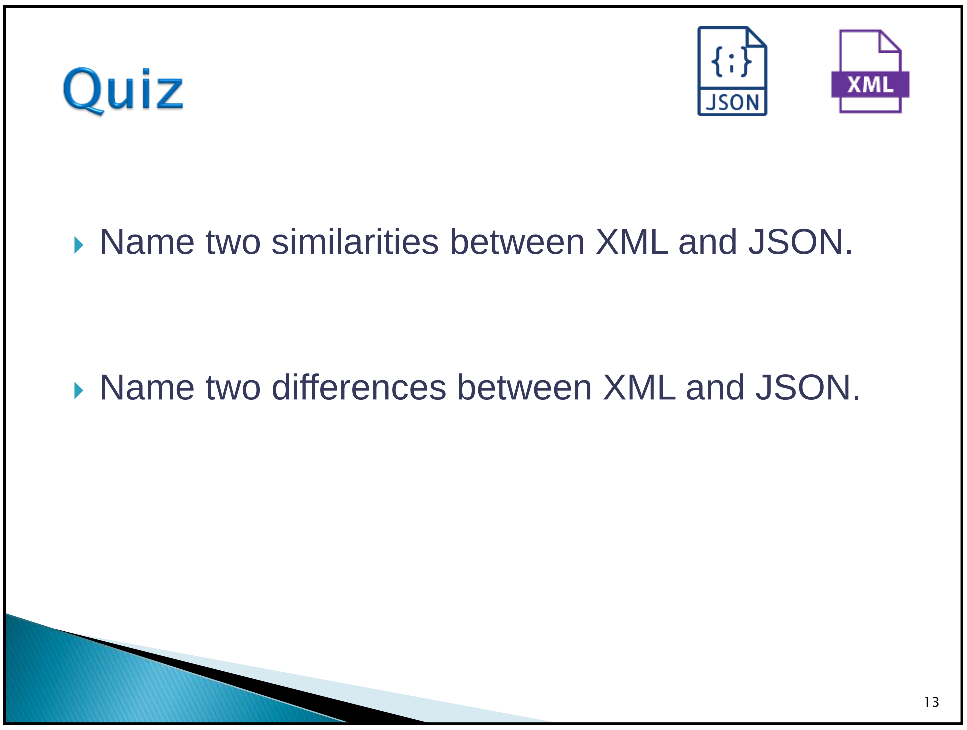
1. XML has more complex constructs, including attributes, namespaces, inheritance and substitution.
2. XML has date data type.
3. XML is useful for both structured and semi-structured data.
4. XML can be either data-centric or document-centric .

JSON

1. JSON has simple structure-defining and document-combining constructs: it lacks attributes, namespaces, inheritance and substitution.
2. JSON, unlike XML (and unlike JavaScript), has no date data type.
   A date is represented in JSON using the available data types, such as string. There are some de facto standards for converting between real dates and strings. But programs using JSON must, one way or another, deal with date representation conversion.
3. JSON is most useful with simple, structured data.
4. JSON is generally data-centric, not document-centric

12

12

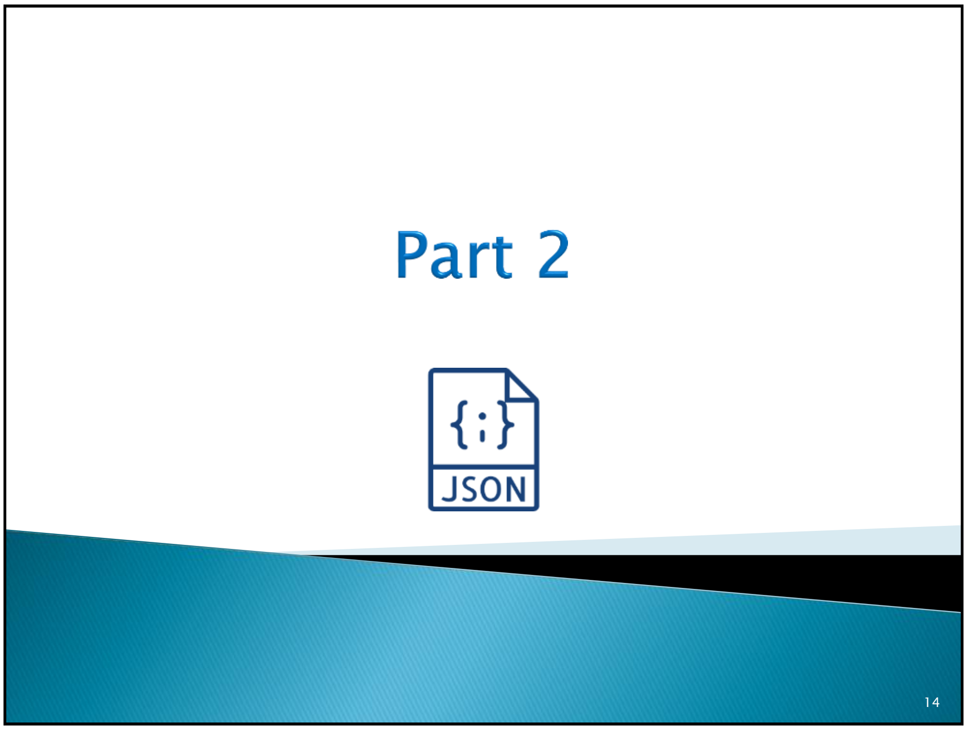


Quiz

- Name two similarities between XML and JSON.
- Name two differences between XML and JSON.

13

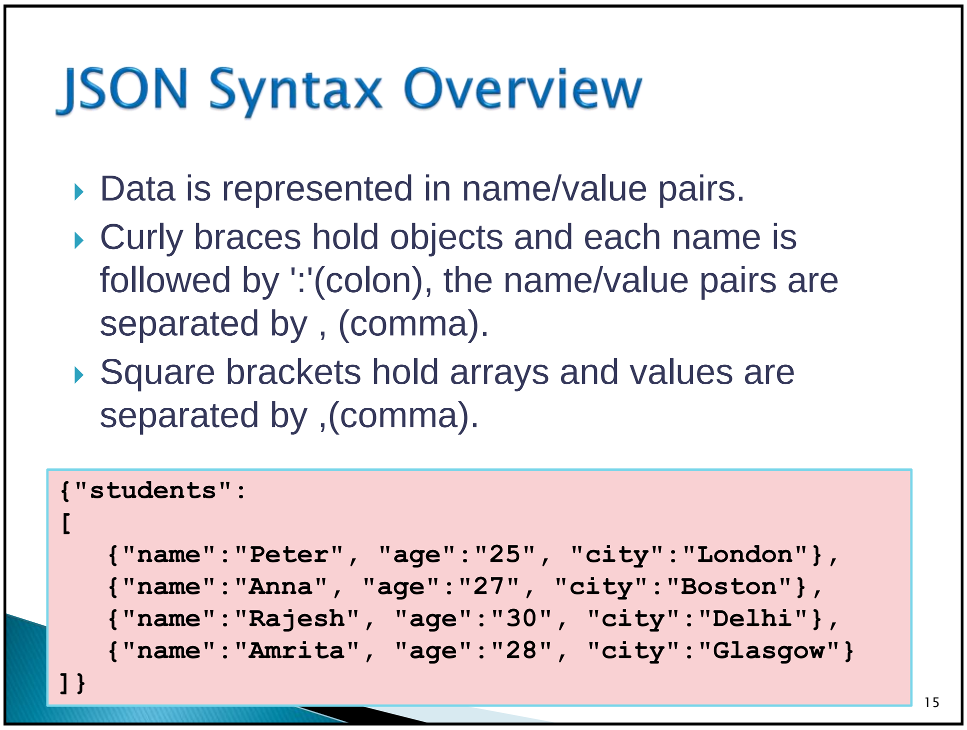
13



Part 2

14

14



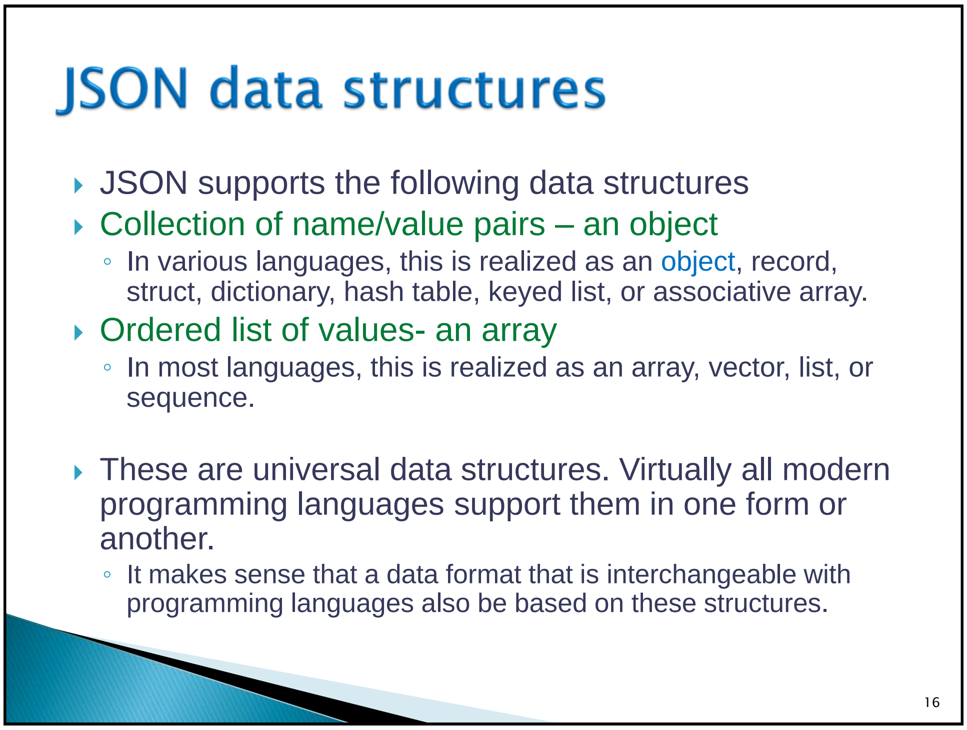
JSON Syntax Overview

- 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).

```
{"students":
[
   {"name":"Peter", "age":"25", "city":"London"},
   {"name":"Anna", "age":"27", "city":"Boston"},
   {"name":"Rajesh", "age":"30", "city":"Delhi"},
   {"name":"Amrita", "age":"28", "city":"Glasgow"}
]}
```

15

15



JSON data structures

- JSON supports the following data structures
- Collection of name/value pairs – an object
  - In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- Ordered list of values- an array
  - In most languages, this is realized as an array, vector, list, or sequence.

- 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.

16

16

JSON data structures – object

- In JSON, an object is an unordered set of name/value pairs.
- An object begins with **{**left brace and ends with **}**right brace.
- Each name is followed by **:**colon and the name/value pairs are separated by **,**comma.
- The keys must be strings and should be different from each other.
- Objects should be used when the key names are arbitrary strings.
- Example:

```
{ "name":"Peter",
   "age":"25",
   "city":"London"
}
```

17

17

JSON objects

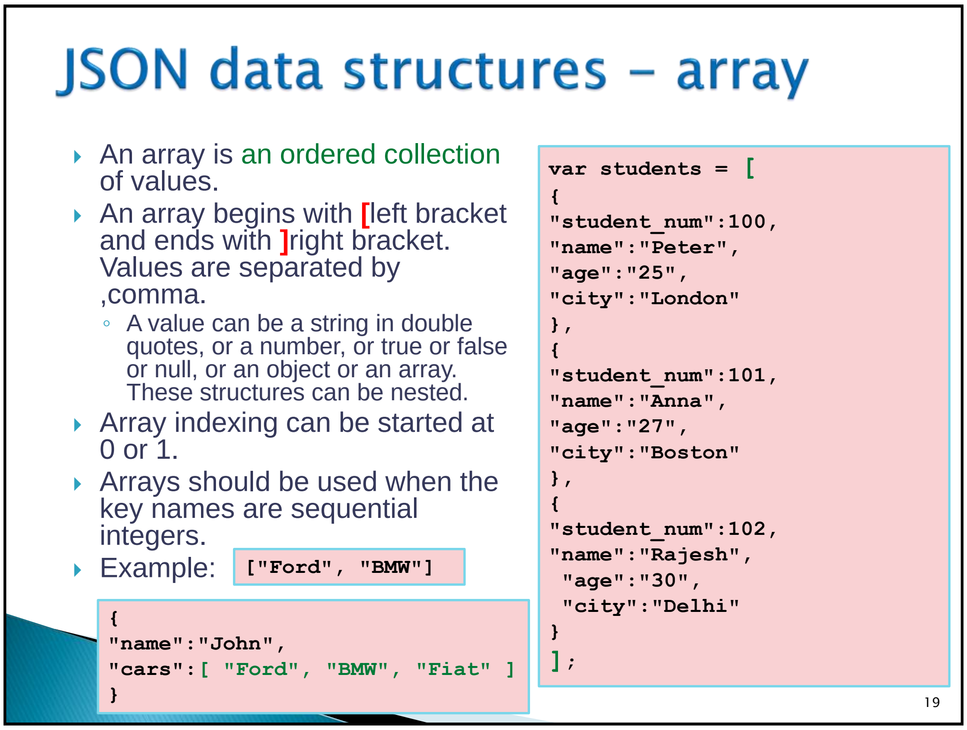
```
var anna = {
  "name" : "Anna Ford",
  "age" : "27",
  "city" : "Boston"
};
```

- This text creates an object that can be accessed using the variable *anna*.
  - Inside an object there can be any number of key-value pairs
- To access the information out of a JSON object use:

```
document.writeln("The name is:  " + anna.name);
document.writeln("her age is: " + anna.age);
document.writeln("her city is: "+ anna.city)};
```

18

18



JSON data structures – array

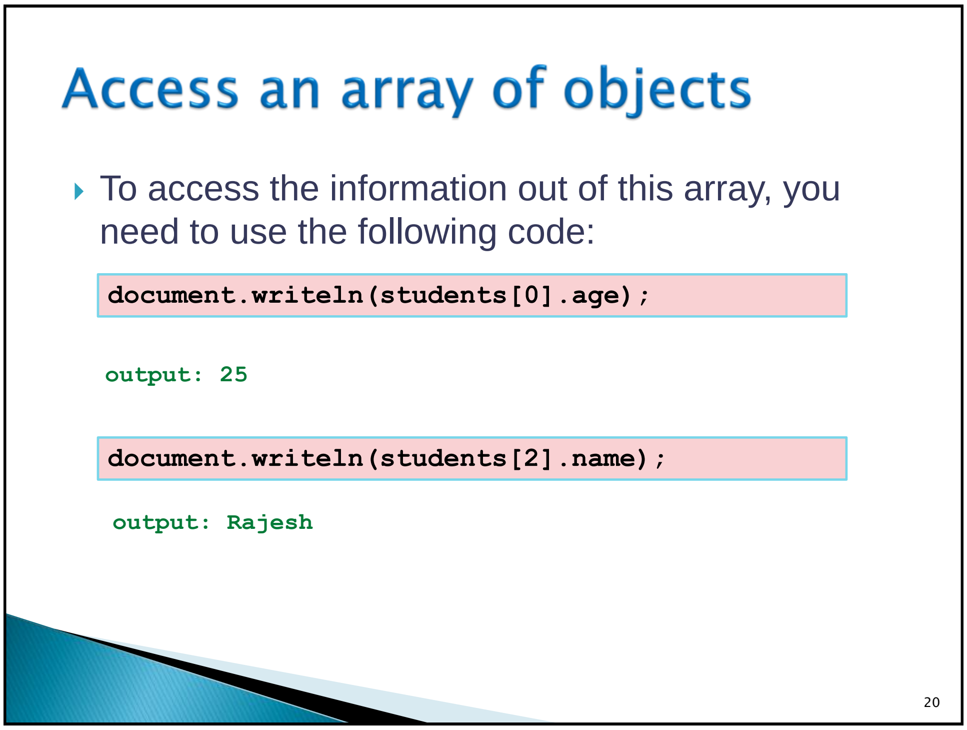
- An array is an ordered collection of values.
- An array begins with [left bracket and ends with]right bracket. Values are separated by ,comma.
  - A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.
- Array indexing can be started at 0 or 1.
- Arrays should be used when the key names are sequential integers.
- Example:

```
["Ford", "BMW"]
```

```
{
"name":"John",
"cars":[ "Ford", "BMW", "Fiat" ]
}
```

```
var students = [
{
"student_num":100,
"name":"Peter",
"age":"25",
"city":"London"
},
{
"student_num":101,
"name":"Anna",
"age":"27",
"city":"Boston"
},
{
"student_num":102,
"name":"Rajesh",
 "age":"30",
 "city":"Delhi"
}
];
```

19



Access an array of objects

- To access the information out of this array, you need to use the following code:

```
document.writeln(students[0].age);
```

```
output: 25
```

```
document.writeln(students[2].name);
```

```
output: Rajesh
```

20

Nesting of JSON objects

- An alternative way of doing the same as above:
- To access:

```
document.writln(students.pete.age);

//output: 25

document.writeln(students.raj.city);

//output: Delhi
```

```
var students = {
  "pete" :
  {
  "name" : "Peter",
  "age" :  "25",
  "city":"London"
  },
 "anna" :
 {
 "name":"Anna",
 "age":"27",
 "city":"Boston
 },
"raj" :
 {
  "name":"Rajesh",
  "age":"30",
  "city":"Delhi"
 }
}
```

Data Types

- **Number:**
  - double- precision floating-point format
    - JSON supports Integer (1-9, 0) and Fraction (5.6)
    - value should not be quoted

```
var obj1 = {mark: 88}
```

- **String**
  - It is a sequence of zero or more **double quoted** Unicode characters

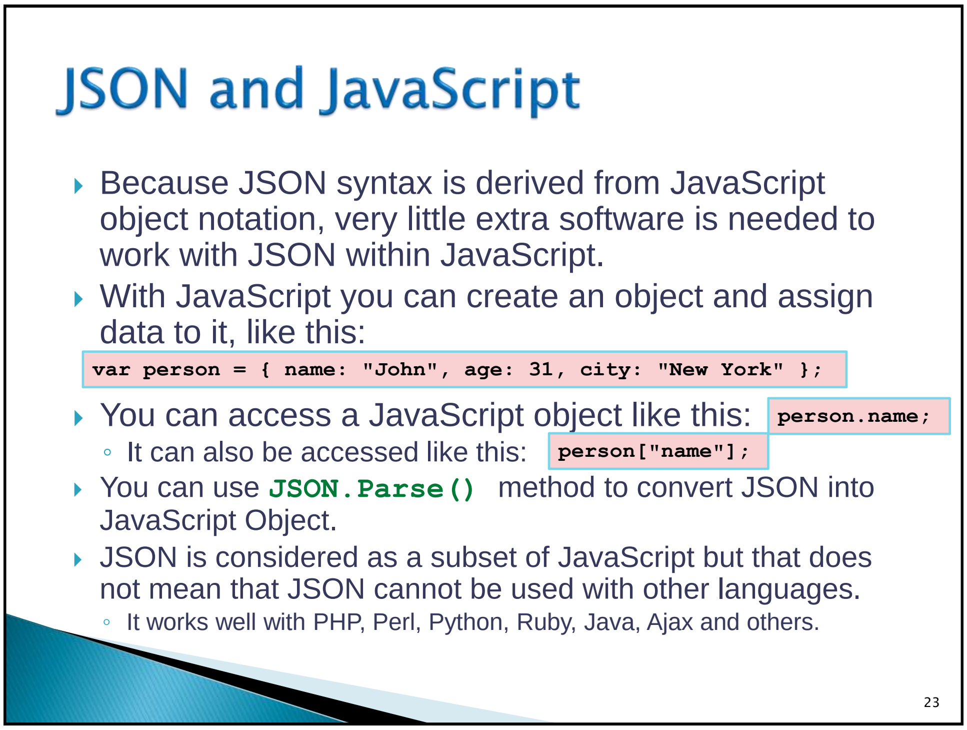
```
var obj2 = {name: "Amrita"}
```

- **Boolean**
  - It includes true or false values:
  - Example

```
var obj3 = {name: "Amrita", mark:88, distinction: true}
```

- **Array**
  - an ordered sequence of values
- **Object**
  - an unordered collection of key:value pairs

Whitespace can be used between any pair of tokens



JSON and JavaScript

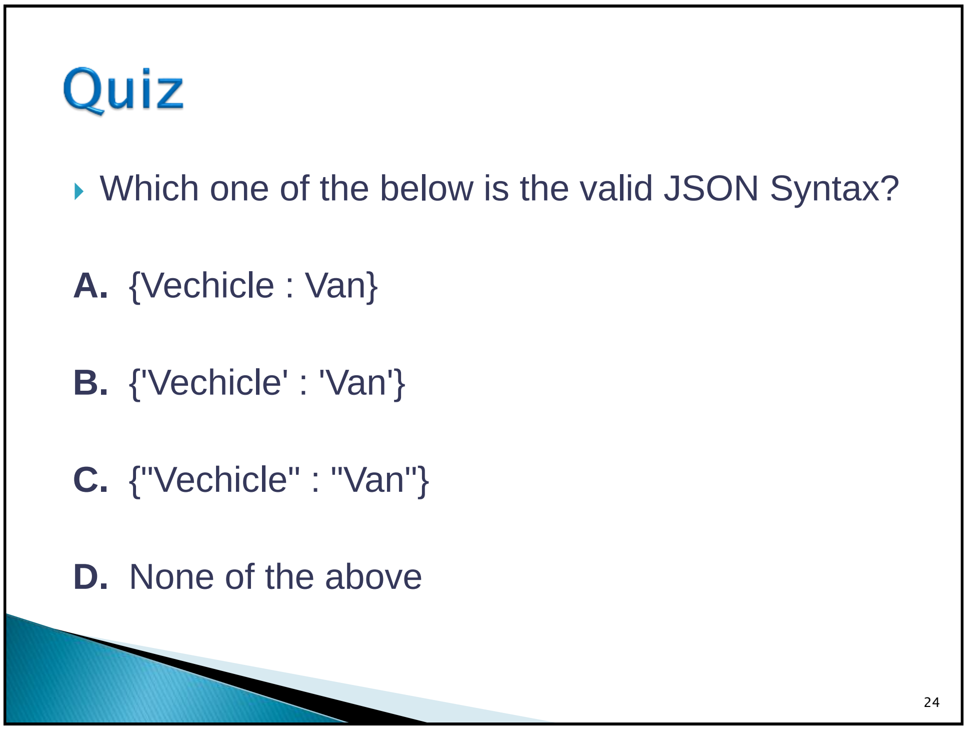
- Because JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.
- With JavaScript you can create an object and assign data to it, like this:

```
var person = { name: "John", age: 31, city: "New York" };
```

- You can access a JavaScript object like this: `person.name;`
  - It can also be accessed like this: `person["name"];`
- You can use **`JSON.Parse()`** method to convert JSON into JavaScript Object.
- JSON is considered as a subset of JavaScript but that does not mean that JSON cannot be used with other languages.
  - It works well with PHP, Perl, Python, Ruby, Java, Ajax and others.

23

23



Quiz

- Which one of the below is the valid JSON Syntax?

A. {Vechicle : Van}

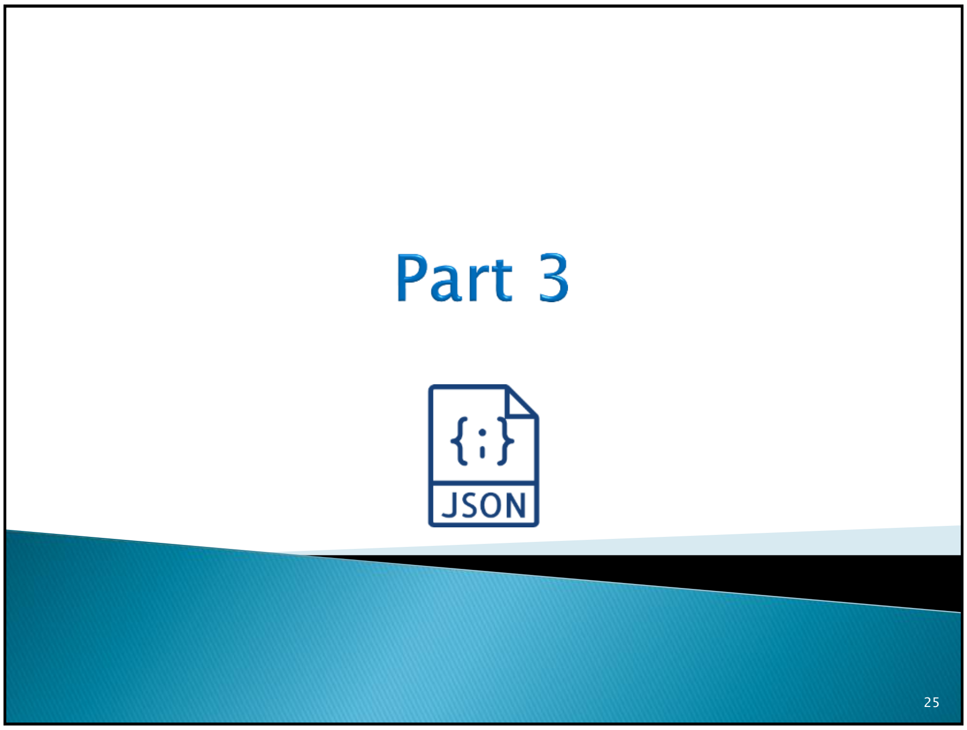
B. {'Vechicle' : 'Van'}

C. {"Vechicle" : "Van"}

D. None of the above

24

24

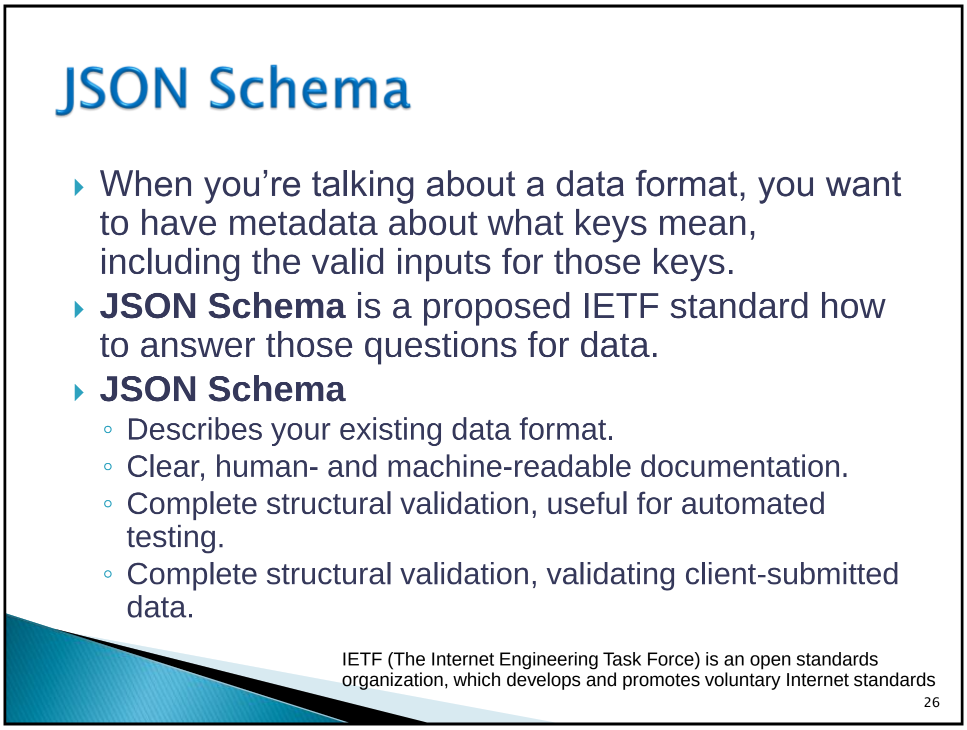


Part 3

JSON

25

25



JSON Schema

- When you're talking about a data format, you want to have metadata about what keys mean, including the valid inputs for those keys.
- **JSON Schema** is a proposed IETF standard how to answer those questions for data.
- **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.

IETF (The Internet Engineering Task Force) is an open standards organization, which develops and promotes voluntary Internet standards

26

26

JSON Schema Keywords

- JSON schema usually starts with four properties, called keywords, which are expressed as JSON keys.
- Schema Keyword: $schema and $id.
  - The $schema keyword states that this schema is written according to a specific draft of the standard and used for a variety of reasons, primarily version control.
  - The $id keyword defines a URI for the schema, and the base URI that other URI references within the schema are resolved against.
- Schema Annotations: title and description.
  - The title and description annotation keywords are descriptive only. They do not add constraints to the data being validated. The intent of the schema is stated with these two keywords.
- Validation Keyword: type.
  - The type validation keyword defines the constraint on JSON data (for example, it has to be a JSON Object).

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://example.com/product.schema.json",
  "title": "Product",
  "description": "A product in the catalog",
  "type": "object"
}
```

7

27

JSON Schema Keywords

- Keyword **`properties`** defines various keys and their value types, minimum and maximum values to be used in JSON file.
- Keyword **`required`** keeps a list of required properties.
- Example:

```
{  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://example.com/product.schema.json",
  "title": "Product",
  "description": "A product in the catalog",
  "type": "object",
   "properties": {
     "id": {
         "description": "The unique identifier for a product",
         "type": "integer"
           },
    "price": {
         "type": "number",
         "minimum": 0,
             },

"required": ["id", "price"]
}
```

28

28

Validation Keywords for Number data type

- **multipleOf**
  - The value of "multipleOf" MUST be a number, strictly greater than 0.
  - A numeric instance is valid only if division by this keyword's value results in an integer.
- **maximum**
  - The value of "maximum" MUST be a number, representing an inclusive upper limit for a numeric instance.
  - If the instance is a number, then this keyword validates only if the instance is less than or exactly equal to "maximum".
- **exclusiveMaximum**
  - The value of "exclusiveMaximum" MUST be number, representing an exclusive upper limit for a numeric instance.
  - If the instance is a number, then the instance is valid only if it has a value strictly less than (not equal to) "exclusiveMaximum".
- **minimum**
  - The value of "minimum" MUST be a number, representing an inclusive lower limit for a numeric instance.
  - If the instance is a number, then this keyword validates only if the instance is greater than or exactly equal to "minimum".
- **exclusiveMinimum**
  - The value of "exclusiveMinimum" MUST be number, representing an exclusive lower limit for a numeric instance.
  - If the instance is a number, then the instance is valid only if it has a value strictly greater than (not equal to) "exclusiveMinimum".

Validation Keywords for Strings

- **maxLength**
  - The value of this keyword MUST be a non-negative integer.
  - A string instance is valid against this keyword if its length is less than, or equal to, the value of this keyword.
  - The length of a string instance is defined as the number of its characters
- **minLength**
  - The value of this keyword MUST be a non-negative integer.
  - A string instance is valid against this keyword if its length is greater than, or equal to, the value of this keyword.
  - The length of a string instance is defined as the number of its characters
  - Omitting this keyword has the same behaviour as a value of 0.
- **pattern**
  - The value of this keyword MUST be a string and this string SHOULD be a valid regular expression
  - A string instance is considered valid if the regular expression matches the instance successfully.

Validation Keywords for Arrays

- **maxItems**
  - The value of this keyword MUST be a non-negative integer; an array instance is valid against "maxItems" if its size is less than, or equal to, the value of this keyword.
- **minItems**
  - The value of this keyword MUST be a non-negative integer; an array instance is valid against "minItems" if its size is greater than, or equal to, the value of this keyword; (default -0).
- **uniqueItems**
  - The value of this keyword MUST be a boolean.
  - If it has boolean value true, the instance validates successfully if all of its elements are unique. (default -false)
- **maxContains**
  - The value of this keyword MUST be a non-negative integer; an array instance is valid against "maxContains" if the number of elements that are valid against the schema for "contains" is less than, or equal to, the value of this keyword.
- **minContains**
  - The value of this keyword MUST be a non-negative integer; an array instance is valid against "minContains" if the number of elements that are valid against the schema for "contains" is greater than, or equal to, the value of this keyword.
  - A value of 0 is allowed, but is only useful for setting a range of occurrences from 0 to the value of "maxContains". A value of 0 with no "maxContains" causes "contains" to always pass validation. (default =1)

31

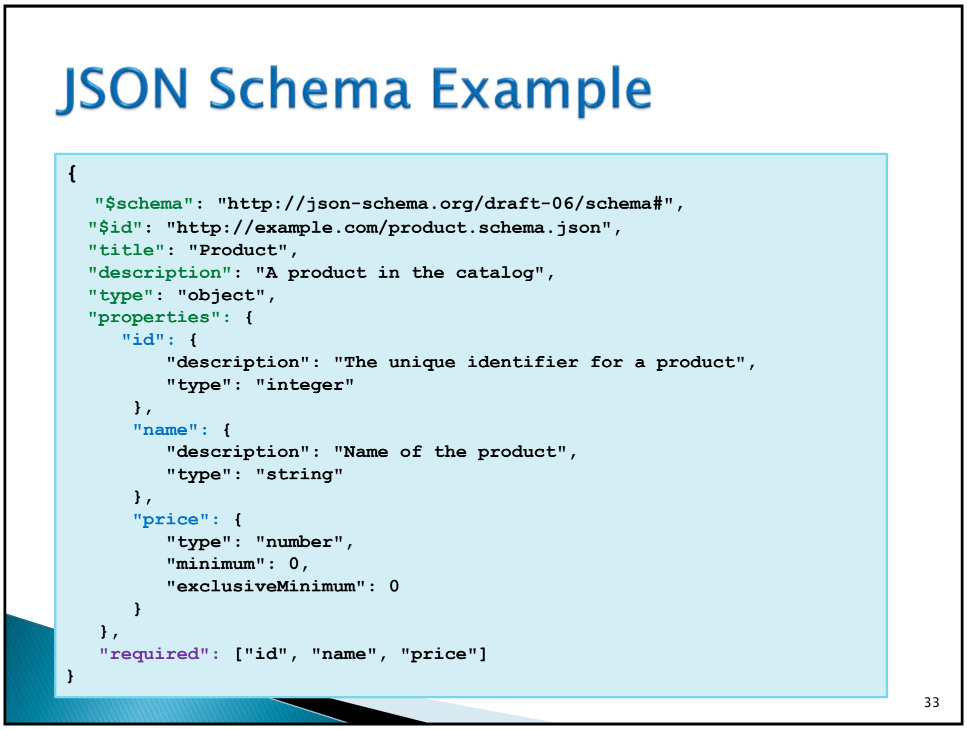
31

Validation Keywords for Objects

- **maxProperties**
  - The value of this keyword MUST be a non-negative integer; an object instance is valid against "maxProperties" if its number of properties is less than, or equal to, the value of this keyword.
- **minProperties**
  - The value of this keyword MUST be a non-negative integer; an object instance is valid against "minProperties" if its number of properties is greater than, or equal to, the value of this keyword.
  - default = 0.
- **required**
  - The value of this keyword MUST be an array. Elements of this array, if any, MUST be strings, and MUST be unique.
  - An object instance is valid against this keyword if every item in the array is the name of a property in the instance; default = an empty array.
- **dependentRequired**
  - The value of this keyword MUST be an object. Properties in this object, if any, MUST be arrays. Elements in each array, if any, MUST be strings, and MUST be unique.
  - This keyword specifies properties that are required if a specific other property is present. Their requirement is dependent on the presence of the other property.
  - Validation succeeds if, for each name that appears in both the instance and as a name within this keyword's value, every item in the corresponding array is also the name of a property in the instance.
  - Omitting this keyword has the same behaviour as an empty object.

32

32

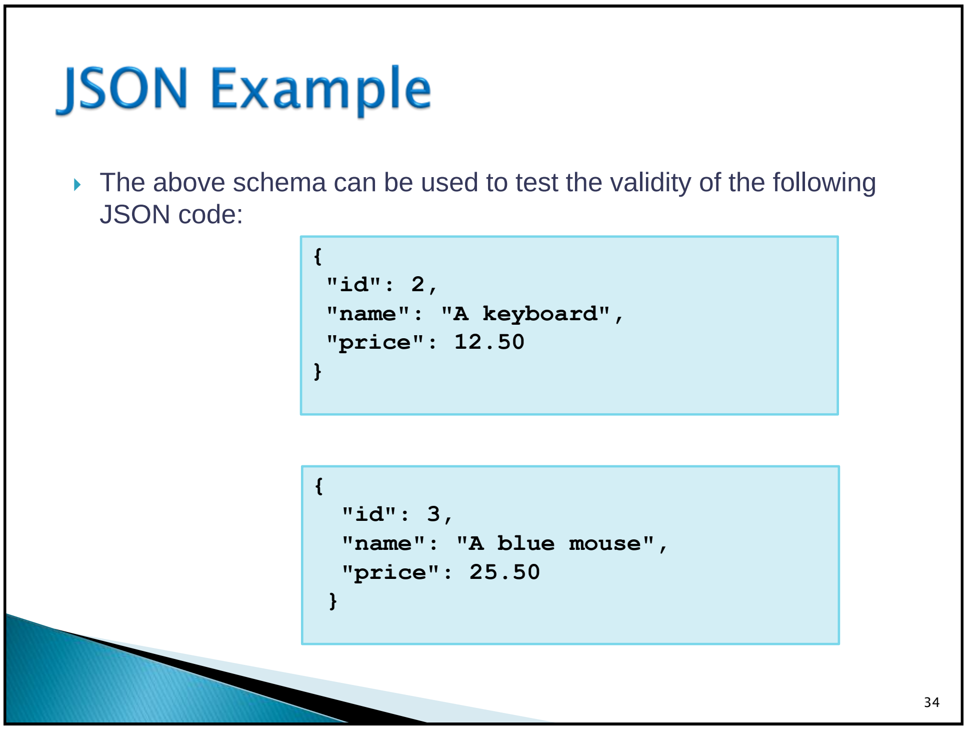


JSON Schema Example

```
{
   "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://example.com/product.schema.json",
  "title": "Product",
  "description": "A product in the 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": 0
      }
   },
   "required": ["id", "name", "price"]
}
```

33

33



JSON Example

- The above schema can be used to test the validity of the following JSON code:

```
{
 "id": 2,
 "name": "A keyboard",
 "price": 12.50
}
```

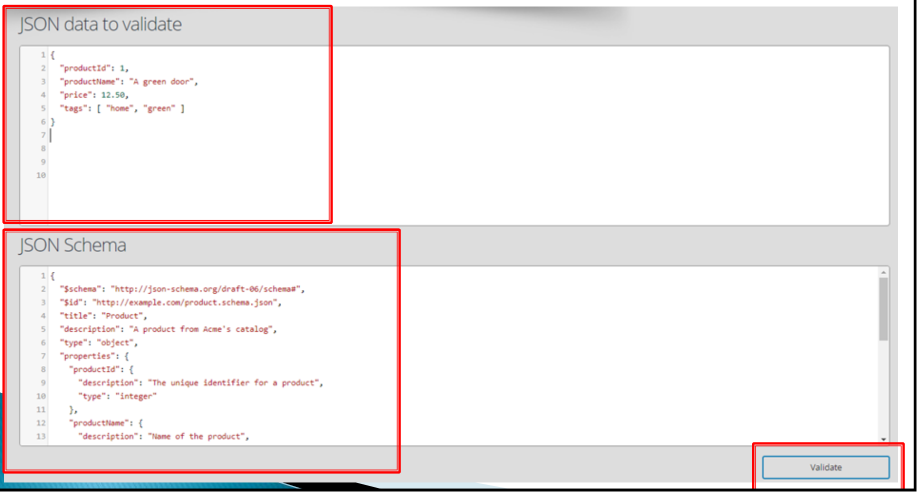
```
{
   "id": 3,
   "name": "A blue mouse",
   "price": 25.50
 }
```

34

34

Validating JSON

- There are many tools available for validating JSON against JSON schema.
- One of them (free) - online JSON Validator: https://www.liquid-technologies.com/online-json-schema-validator



```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.50,
  "tags": [ "home", "green" ]
}
```

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://example.com/product.schema.json",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "productId": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },
    "productName": {
      "description": "Name of the product",
```

35

Quiz

- Is this JSON valid against the above schema?

```
{
   "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://example.com/product.schema.json",
  "title": "Product",
  "description": "A product in the catalog",
   "type": "object",
   "properties": {
   "id": {
         "description": "The unique id for a prod",
         "type": "integer"
         },
   "name": {
          "description": "Name of the product",
          "type": "string"
          },
   "price": {
          "type": "number",
          "minimum": 0,
          "exclusiveMinimum": 0
       }
    },
    "required": ["id", "name", "price"]
}
```

```
[
   {
      "id": "four",
      "name": "USB",
      "price": 12.50
   },

   {
      "name": "Monitor",
      "id": "five",
      "price": 125.50
   }
]
```

36

36

Further reading

- Information about JSON: http://www.json.org
- Definition of the JSON Data Interchange Format: http://www.ecma-international.org/publications/standards/Ecma-404.htm
- Information about JSON Schema: https://json-schema.org/draft/2019-09/json-schema-validation.html
- ECMAScript Language Specification: http://www.ecma-international.org/publications/standards/Ecma-262.htm

Essentials

- ✓ Introduced JSON
- ✓ Compared JSON and XML
- ✓ Learned JSON syntax
- ✓ Discussed JSON data structures
- ✓ Introduced JSON schema