# CSE 154

LECTURE 24: JSON

# Pros and cons of XML

- pro:
  - standard open format; don't have to "reinvent the wheel" for storing new types of data
  - can represent almost any general kind of data (record, list, tree)
  - easy to read (for humans and computers)
  - lots of tools exist for working with XML in many languages
- con:
  - bulky syntax/structure makes files large; can decrease performance (example)
  - can be hard to "shoehorn" data into a good XML format
  - JavaScript code to navigate the XML DOM is bulky and generally not fun

# An example of XML data

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note private="true">
  <from>Alice Smith (alice@example.com)</from>
  <to>Robert Jones (roberto@example.com)</to>
  <to>Charles Dodd (cdodd@example.com)</to>
  <subject>Tomorrow's "Birthday Bash" event!</subject>
  <message language="english">
    Hey guys, don't forget to call me this weekend!
  </message>
</note>
```

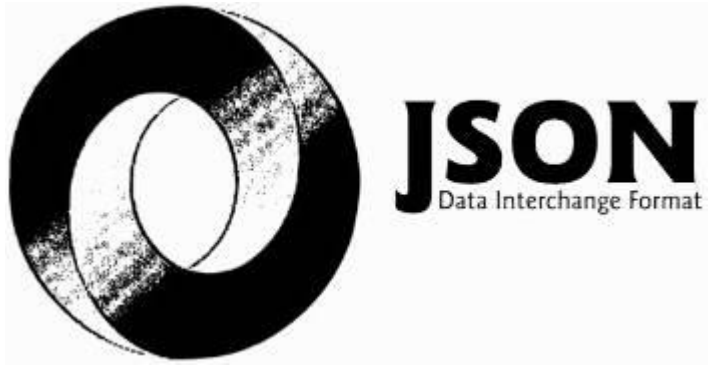
XML

- fairly simple to read and understand
- can be parsed by JavaScript code using XML DOM
- Is there any other data format that is more natural for JS code to process?

# JavaScript Object Notation (JSON)

**JavaScript Object Notation (JSON):** Data format that represents data as a set of JavaScript objects

- invented by JS guru Douglas Crockford of Yahoo!
- natively supported by all modern browsers (and libraries to support it in old ones)
- not yet as popular as XML, but steadily rising due to its simplicity and ease of use




# Background: Creating a new object

```
var name = {
  fieldName: value,
  ...
  fieldName: value
};
```
JS

```
var pt = {
  x: 4,
  y: 3
};
pt.z = -1;
alert("(" + pt.x + ", " + pt.y + ", " + pt.z + ")");   // (4, 3, -1)
```

- in JavaScript, you can create a new object without creating a class
- you can add properties to any object even after it is created (`z`)

# More about JavaScript object syntax

```
var person = {
  name: "Philip J. Fry",                          // string
  age: 23,                                        // number
  "weight": 172.5,                                // number
  friends: ["Farnsworth", "Hermes", "Zoidberg"],  // array
  getBeloved: function() { return this.name + " loves Leela"; }
};
alert(person.age);                          // 23
alert(person["weight"]);                    // 172.5
alert(person.friends[2]));                  // Zoidberg
alert(person.getBeloved());                 // Philip J. Fry loves Leela
```

- an object can have methods (function properties) that refer to itself as `this`
- can refer to the fields with `.`*fieldName* or `["`*fieldName*`"]` syntax
- field names can optionally be put in quotes (e.g. `weight` above)

# Repeated: Example XML data

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note private="true">
  <from>Alice Smith (alice@example.com)</from>
  <to>Robert Jones (roberto@example.com)</to>
  <to>Charles Dodd (cdodd@example.com)</to>
  <subject>Tomorrow's "Birthday Bash" event!</subject>
  <message language="english">
    Hey guys, don't forget to call me this weekend!
  </message>
</note>
```

XML

- Could we express this message data as a JavaScript object?
- Each attribute and tag could become a property or sub-object within the overall message object

# The equivalant JSON data

```
{
  "private": "true",
  "from": "Alice Smith (alice@example.com)",
  "to": [
    "Robert Jones (roberto@example.com)",
    "Charles Dodd (cdodd@example.com)"
  ],
  "subject": "Tomorrow's \"Birthday Bash\" event!",
  "message": {
    "language": "english",
    "text": "Hey guys, don't forget to call me this weekend!"
  }
}
```

JSON

# Valid JSON

```
var student = {                                  // no variable assignment
  "first_name": 'Bart',                          // strings must be double-quoted
  last_name: "Simpson",                          // property names must be quoted
  "birthdate": new Date("April 1, 1983"),        // Date objects not supported
  "enroll": function() {                         // Functions not supported
    this.enrolled = true;
  }
};
```

JSON

- JSON has a few rules that differ from regular JS:
  - Strings must be quoted (in JS, single- or double-quoted are allowed)
  - All property/field names must be quoted
  - Unsupported types: `Function`, `Date`, `RegExp`, `Error`
  - All others supported: `Number`, `String`, `Boolean`, `Array`, `Object`, `null`
- Numerous validators/formatters available: JSONLint, JSON Formatter & Validator, Free Formatter, JSON Validator

# Browser JSON methods

| method | description |
|---|---|
| JSON.parse(*string*) | converts the given string of JSON data into an equivalent JavaScript object and returns it |
| JSON.stringify(*object*) | converts the given object into a string of JSON data (the opposite of JSON.parse) |

- you can use Ajax to fetch data that is in JSON format
- then call `JSON.parse` on it to convert it into an object
- then interact with that object as you would with any other JavaScript object

# JSON expressions exercise

Given the JSON data at right, what expressions would produce:

- The window's title? *(use the Console)*
- The image's third coordinate?
- The number of messages?
- The y-offset of the last message?

```
var data = JSON.parse(this.responseText);
```

```
{
  "window": {
    "title": "Sample Widget",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "images/logo.png",
    "coords": [250, 150, 350, 400],
    "alignment": "center"
  },
  "messages": [
    {"text": "Save", "offset": [10, 20]},
    {"text": "Help", "offset": [ 0, 50]},
    {"text": "Quit", "offset": [30, 15]}
  ],
  "debug": "true"
}
```

JSON

```
var title = data.window.title;
var coord = data.image.coords[2];
var len = data.messages.length;
var y = data.messages[len - 1].offset[1];
```

# JSON example: Books

Suppose we have a service books_json.php about library books.

- If no query parameters are passed, it outputs a list of book categories:

```
{ "categories": ["computers", "cooking", "finance", ...] }
```
JSON

- Supply a category query parameter to see all books in one category:
  http://webster.cs.washington.edu/books_json.php?category=cooking

```
{
  "books": [
    {"category": "cooking", "year": 2009, "price": 22.00,
     "title": "Breakfast for Dinner", "author": "Amanda Camp"},
    {"category": "cooking", "year": 2010, "price": 75.00,
     "title": "21 Burgers for the 21st Century", "author": "Stuart Reges"},
    ...
  ]
}
```
JSON

# JSON exercise

Write a page that processes this JSON book data.

- Initially the page lets the user choose a category, created from the JSON data.
  - ○ Children ○ Computers ○ Finance List Books

- After choosing a category, the list of books in it appears:

Books in category "Cooking":

- Breakfast for Dinner, by Amanda Camp (2009)
- 21 Burgers for the 21st Century, by Stuart Reges (2010)
- The Four Food Groups of Chocolate, by Victoria Kirst (2005)

# Working with JSON book data

```
function showBooks() {
  // add all books from the JSON data to the page's bulleted list
  var data = JSON.parse(this.responseText);
  for (var i = 0; i < data.books.length; i++) {
    var li = document.createElement("li");
    li.innerHTML = data.books[i].title + ", by " +
        data.books[i].author + " (" + data.books[i].year + ")";
    document.getElementById("books").appendChild(li);
  }
}
```

JS

# Bad style: the eval function

```
// var data = JSON.parse(this.responseText);
var data = eval(this.responseText);   // don't do this!
...
```
JS

- JavaScript includes an `eval` keyword that takes a string and runs it as code
- this is essentially the same as what `JSON.parse` does,
- but `JSON.parse` filters out potentially dangerous code; `eval` doesn't
- `eval` is evil and should not be used!