Theme 8

Ajax and Json

XML in Practice

Reminder:

- You use XML to create **custom markup languages**…
- …each with a **specific purpose**.

eg:
google maps
word doc

- In the final themes, we'll look at…
- …examples of XML as it is used in practice…
- …similar technology that serve the same purpose.

english was created to speak to each other, number would be hard (xml vs json)

JSON

- **JSON** = **J**ava**S**cript **O**bject **N**otation
- It is **based** on JavaScript syntax but it is **not** JavaScript
- It is a syntax for **serializing** objects, arrays, numbers, strings, booleans, and null.
- JSON is a **lightweight data-interchange format**.
  - It is an alternative to XML
  - http://www.json.org/xml.html

Why JSON

- Easy for humans to **read** and **write**.
- Machines can easily **parse** and generate it.
- It is based on a **subset of JavaScript**.
- It is completely independent language but uses conventions that are familiar to programmers of the C-family languages.

JSON

- It is an international standard (as of 2013)

- It is also very popular for a number of reasons:
  - The explosive growth of **RESTful APIs** based on JSON
  - The simplicity of JSON's basic data structures
  - The increasing popularity of JavaScript

- JSON fits well into JS development

JSON

- JSON is built on 2 universal data structures:
  - A collection of name/value pairs (**object**).
  - An ordered list of values (**array**).

- You can validate JSON here: https://jsonlint.com/

- You can also find an online JSON editor (with some nice features) here: https://jsoneditoronline.org/

JSON Object

- JSON work off a key/value pair system

```
{"firstname": "Daddy"}
```

- Use double quote around both **KEY** and **VALUE**

- Like XML, JSON file needs to be valid

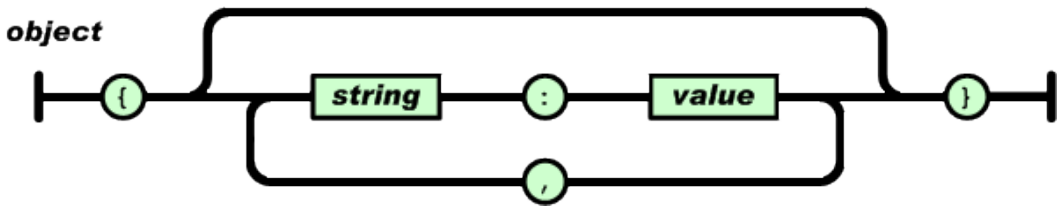
JSON representation: Object

The object is an unordered set of name/value pairs.

The object begins with { and closes with }.

Each name is followed by a : (colon) and the name/value pairs are separated by a , (comma).

```
{
    "firstname": "Daddy",
    "surname": "Long Legs",
    "age": 98
}
```



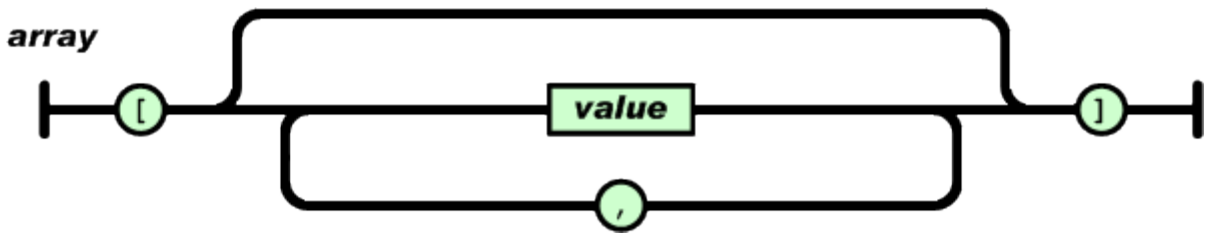
JSON representation: Array

An array is an ordered collection of values.

An array begins with [and ends with].

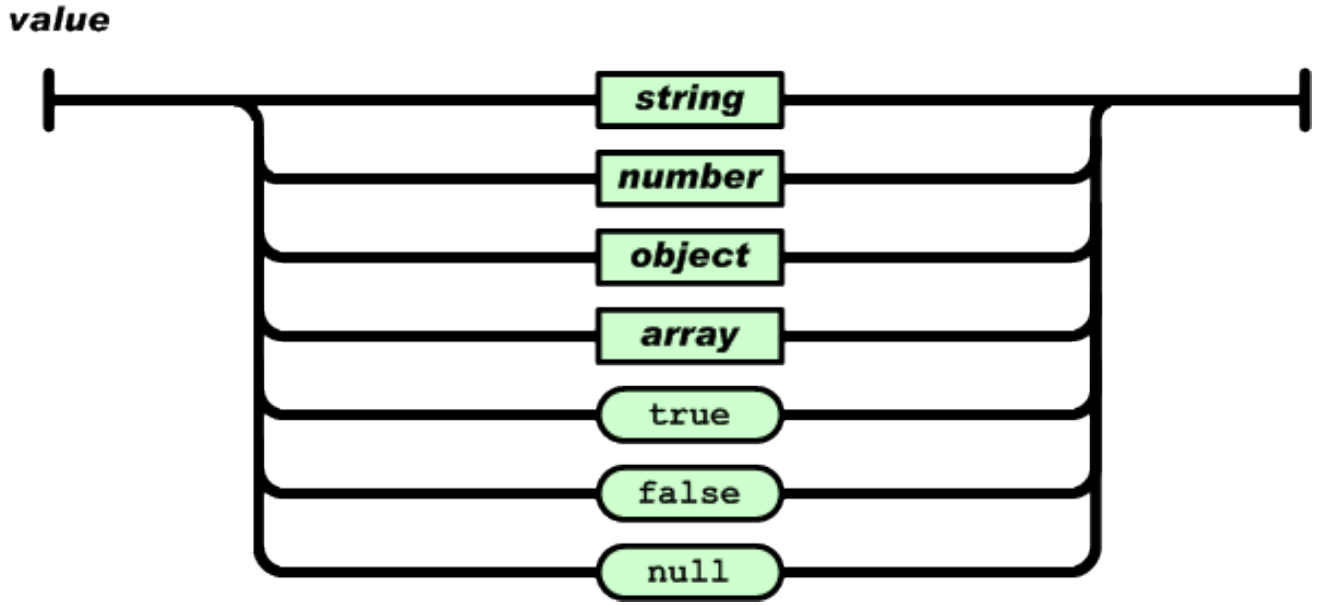
The values are separated by a , (comma).

```
{
    "user01": [
        "one", "two", "three"
    ]
}
```



JSON representation: Value

- A value can be:
  - a string in **double quotes**; single quotes are forbidden
  - a number;
  - boolean
  - null
  - an object;
  - an array.



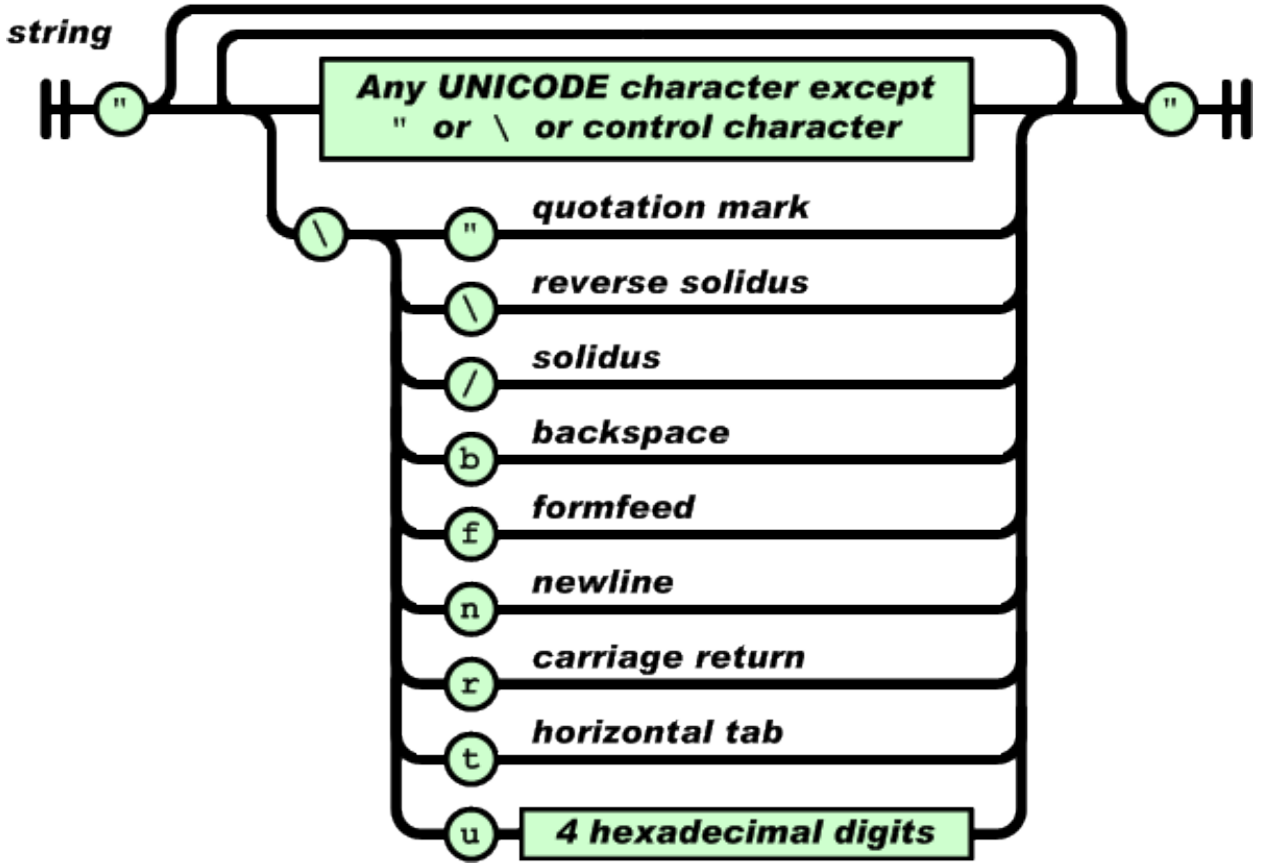
JSON representation: Value

These structures can be nested.

```
{
    "dataOne": ["one", "two", "three"],
    "dataTwo": {
        "type01": "theString",
        "type02": 12
    },
    "dataThree": "theString again",
    "dataFour": 21
}
```

JSON representation: String

- String is a:
  - Sequence of zero or more Unicode characters…
  - wrapped in double quotes, using backslash escapes.



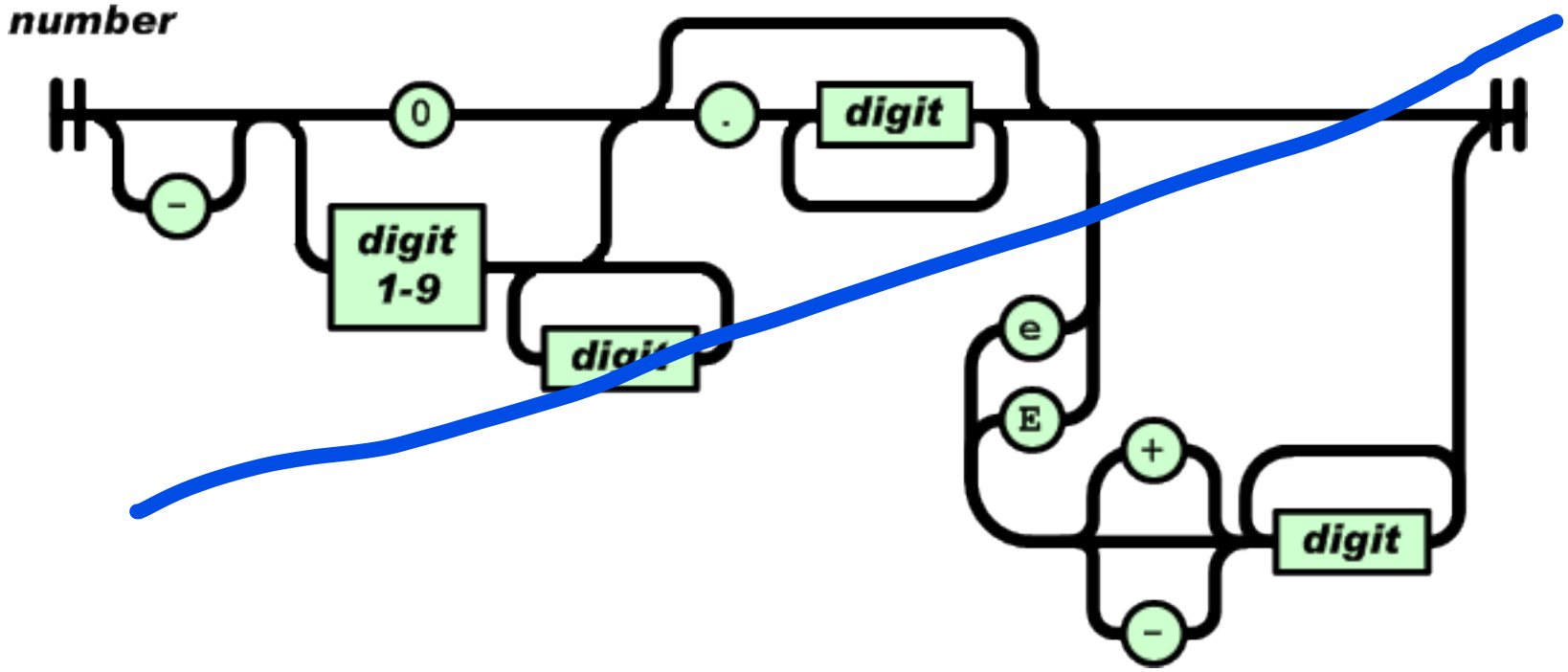
JSON representation: String

Backslash-escaped characters are used for special characters

```
{
      "quote": "H.L. Mencken said: \"For every complex problem
      there is an answer that is simple, clear and wrong.\""
}
```

JSON representation: Number

- A number is represented in the same way in Java or C, **but** octal and hexadecimal formats are not used



Don’t comment in JSON

JSON files don’t allow for comments.

Douglas Crockford (the creator of JSON) initially allowed for comments in JSON files, but removed them, because:

- “*He believed that comments weren’t useful.*
- *JSON parsers had difficulties supporting comments.*
- *People were abusing comments. Removing comments simplified and enabled cross-platform JSON support.”*

There will probably **never be a newer version** of the core JSON standard, again largely because of **interoperability** and backwards **compatibility**

JSON file

- The official Internet media type for JSON is **application/json**.

- The JSON filename extension is **.json**.
  - e.g., data.json:

```
{
    "name": "George",
    "age": 29,
    "friends": ["John ", "Sarah", "Albert"]
}
```

Reading JSON files

- The most common way to read a **valid JSON file** is to use jQuery (AJAX) functions like $.ajax or $.getJSON.

- The alternative is to save your file with a **.jsonp** extension, so that you can include **JS code** in the file. This is not recommended since it defeats the purpose of JSON by not using a **platform-independent** JSON file.

Working with JSON

- Code to **generate** and **parse** JSON-format data is available in many programming languages.

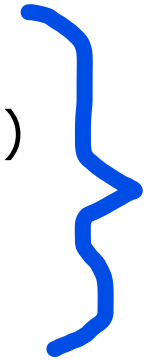
- In JS it can be done using one of two **JSON object methods**.

  JSON.parse()

  One for parsing JavaScript Object Notation (JSON)

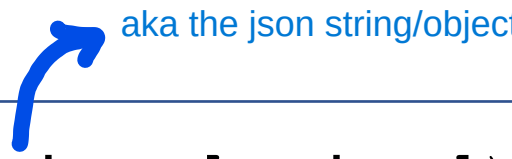
  JSON.stringify()

  One for converting values to JSON



native to javascript

Receiving JSON




```
parse(string, [reviver])
```

Converts a **JSON string** into a **JavaScript object**

string

- This is the JSON string being converted.

[reviver]

- It can optionally transform the data before returning it.

- Note: We are referring to the **JSON object** as a **JSON string**…
  …because JS sees the object as but a string of character…
  …the term JSON objects means nothing to JS

Example: Parse

```
let jsonstr = '{"name":"George","age":29,"friends":["John",
"Sarah", "Albert"]}';
let george = JSON.parse(jsonstr); //convert JSON string to object
alert(george.age); //alerts 29
```

```
1  JSON.parse('{}');              // {}
2  JSON.parse('true');            // true
3  JSON.parse('"foo"');           // "foo"
4  JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
5  JSON.parse('null');            // null
```

Accessing Data in JSON

```
document.writeln(employees.sales[0].firstName);
//shows the first person from the sales array

document.writeln(employees.accounting[1].firstName);
//shows the second person in the accounting array

document.writeln(employees.sales[0]["firstName"]);
//shows the first person from the sales array
```

Native JSON parser

[reviver] is an **optional** parameter:

A **user defined function** that makes further changes to the JSON object.

The function is applied **recursively** to every member and replaces each one with the value returned by the function.

uses for reviver?
- date
- currency

Example: Reviver A

```
let JSONobj = JSON.parse('{"p": 5}', (key, value) => {
   if(typeof value === 'number'){
         return value * 2;
      }
   return value;
     });

console.log(JSONobj);
// { p: 10 }
```

uses for reviver?
- date
- currency

Example: Reviver B

print: 1,2,4,6,5,3

```
JSON.parse('{"1": 1, "2": 2, "3": {"4": 4, "5": {"6": 6}}}',
    (key, value) => {
            console.log(key);
    // log the current property name, the last is "".
            return value;
    });
```

JSON Object to string

```
stringify(obj, [replacer], [space])
```

Converts a **JS object** into a **JSON syntax string**

obj

- It takes a JS object and returns a JSON syntax string

[replacer]

- It can optionally transform the data before returning it.

[space]

- It can optionally add data before returning it.

Example: Stringify

```
let jsobj = {
    name: "George",
    age: 29,
    friends: ["John", "Sarah", "Albert"]
};

let jsonstr = JSON.stringify(jsobj);
alert(typeof jsonstr); //string
```

Stringify replacer

- **[replacer]** is an **optional** parameter:
  - Can be either a **function** or an **array** of String/Number objects.
  - It steps through each member within the JSON object to let you decide what value each member should change to.

Stringify replacer

As a **function** it can return:

A **number**, **string**, or **Boolean**, which replaces the property's original value with the returned one.

An **object**, which is serialized then returned. Object methods or functions are not allowed, and are removed instead.

Null, which causes the property to be removed.

As an **array**,

the values defined inside it corresponds to the names of the properties inside the JSON object that should be retained when converted into a JSON object.

Example: Stringify Replacer

```
let jsonstr = JSON.stringify(jsonobj, (key, value) => {
if(typeof value === "number")
      return value + 1;
return value;
});

console.log(jsonstr);
//OUTPUT: '{"name":"George", "age":30, "friends":["John",
"Sarah", "Albert"]}'
```

Stringify space

help readability

[space] is an **optional** parameter:

A **Number** object:

inserts x number of white space (up to 10) into the JSON string before each name/value pair for readability.

A **String** object:

inserts the specified string (up to 10 characters) into the JSON string before each name/value pair for readability.

Example: Stringify and Reviver

We store a **date** inside an object property:

```
let jobduty = {   "thedate": new Date().getTime(),
                  "thetask": "Take out garbage"            };
```

We convert the object to a JSON string and then back to an object:

```
let jobstr = JSON.stringify(jobduty);
let jobjson = JSON.parse(jobstr);
alert(`${jobjson.thedate} ${typeof jobjson.thedate}`);
//1535113573822 number
```

The date will not be a date object but an actual number (date in milliseconds).

eg:
other practical reasons to reviver?

Example: Stringify and Reviver

- By using a **reviver function**, we can change that so the property returns an actual date representation of the date:

```
let jobstr = JSON.stringify(jobduty);
let jobjson = JSON.parse(jobstr, (key, value) => {
    if(key === "thedate") //if "thedate" property
          return new Date(value);
    else
          return value;
});
alert(`${jobjson.thedate} ${typeof jobjson.thedate}`);
//Tue Mar 09 2010 00:02:23 GMT-0800 (Pacific Standard Time)
object
```

Parsing JSON in PHP

- To handle JSON in PHP, there is a JSON extension available.

- Two functions:
  - **json_encode();**
  - **json_decode();**

- Useful converting and parsing JSON data through PHP

Parsing JSON in PHP

- Creating JSON with PHP array:

```
$json_data = array(
    "id" => 1,
    "name" => "mike",
    "country" => "usa",
    "office" => array("microsoft", "oracle")
);
echo json_encode($json_data);
```

Parsing JSON in PHP

- Decode JSON with PHP array:

```
$json_string = '{
   "id": 1,
   "name": "mike",
   "country": "usa",
   "office": ["microsoft", "oracle"]
}';
$obj = json_decode($json_string);
print $obj->{'name'};
```

XML object conversation

- Convert XML Object to JSON

```
$xml = simplexml_load_string($xml_string);
$xml = simplexml_load_file($xml_filename);

$json = json_encode($xml);
//Take it further and convert the object to a PHP array
$array = json_decode($json,TRUE);
```

AJAX

- Today, XML is also often used in **AJAX**.

- **AJAX** = **A**synchronous **J**avaScript **a**nd **X**ML.

- **The purpose of AJAX :**
  - To allow a functions to run concurrently...
  - ...**without** refreshing the page in the browser.

AJAX

- AJAX itself is **not** a language, but a **technique**.

- It’s a specific way of using existing languages together.

- **AJAX involves:**
  - **HTML** & **CSS** to display the web page;
  - **XML** & **XSLT** to exchange data between server and browser;
  - **JavaScript** to facilitate the process programmatically.

don not have to be hosted, AJAX works on client side

AJAX

- AJAX works with the **XMLHttpRequest** object.

```
xhr = new XMLHttpRequest();
```

- It allows for **asynchronous** client/server communication.

- **Common examples of AJAX in use:**
  - Google’s search results that appear as you type;
  - Your Facebook status that appears on your feed as you post it.

How to AJAX

- **XMLHttpRequest** provides two methods:
  - **Open**: Creates a connection
  - **Send**: Send a request

- The function will then return data in **XML** or **plain text**

- Along with the data ready state value will also be attached
  - 0: not initialized.
  - 1: connection established.
  - 2: request received.
  - 3: answer in process.
  - 4: finished.

How to AJAX

Step 1: **Create** an Instance
Create an XMLHttpRequest()

Step 2: **Wait** for a response
Check the readyState

Step 3: **Make** the request
open or send

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange =
function() {
instructions to process the
response
};
if (xhr.readyState == 4)
xhr.open('GET',
'http://www.xul.fr/somefile.
xml',
true);
xhr.send(null);
```

JSON and AJAX

Developers use JSON to pass AJAX updates between the client and the server

Any data that is updated using AJAX can be stored using the JSON format on the web server.

AJAX is used so that **javascript** can retrieve these JSON files

Generally these operation is used to achieve one of two things:

- **Store** data for further processing before displaying
- Assign the data to the DOM element in the webpage and **display** them

JSON in context

- Easiest way of using JSON data in a web environment using jQuery (covered in IMY 220)

```
$.ajax(
    type:'GET',
    url:"http://example.com/users/feeds/",
    data:"format=json&id=123",
    success:function(feed) {
        document.write(feed);
    },
    dataType:'jsonp'
);
```