# THE // FLATIRON SCHOOL

## JSON & AJAX

- What is JSON?
- What is AJAX?

## JSON

What is JSON?

JSON stands for Javascript Object Notation. The format was specified by Douglas Crockford as a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for many languages.

## JSON

Why use it?

JSON allows us to package object data into a format we can pass to other languages. It is kind of like packaging up the code for shipping in a convenient format.

## JSON

```
// Let's imagine you've purchased some furniture from
a store, and you want it delivered. In the shop, the
chest-of-drawers you've purchased is a living object:
```

```
var chestOfDrawers = {
    color: 'red',
    numberOfDrawers: 3
}
```

## JSON

```
// It’s easier to ship if the company dismantles it
(convert from Object to JSON using JSON.stringify()).

chestOfDrawers = JSON.stringify(chestOfDrawers);
```

## JSON

```
// Now, it’s in flat pack form easier to transport. If
checked the value it would now appear like this:
```

```
'{"color":"red","numberOfDrawers":3}'
```

## JSON

```
'{"color":"red","numberOfDrawers":4}'

// Notice in JSON all properties get  double quotes
and strings also get double quotes. In fact single
quotes are forbidden. Numbers and Boolean do not
neccesarily need quotes. All other syntax rules follow
the same as Javascript Object notation rules.
```

## JSON

```
// (To receieve/get the furniture we can use
$getJSON();) When you receive it, you can rebuild the
chest-of-drawers (using $.parseJSON();).


chestOfDrawers = $.parseJSON(chestOfDrawers);
```

## JSON

```
// Its now back in an object literal form.
```

The reason behind JSON/ XML and YAML is to enable data to be transferred between programming languages in a format both participating languages can understand; you can't give PHP or C++ your JavaScript object directly; because each language represents an object differently under-the-hood. However, because we've stringified the object into JSON; i.e. a standardized way to represent data, we can transmit the JSON representation of the object to another langauge (C++, PHP, Ruby, Python), they can recreate the JavaScript object we had into their own object based on the JSON representation of the object.

```
      */
```

## JSON

```
// Are there any limitations when creating JSON?

// Yes, unfortunately we will lose methods of an
object when converting into JSON.

var person = {
    name: 'Fred', age: 22,
    birthday: function() { this.age += 1; }
}
```

## JSON

```
// Let’s try

person = JSON.stringify(person);

// Creates

'{"name":"fred","age":22}'

//The birthday method is gone! Oh my! So JSON is
really meant for passing data and not actions or
programming logic.
```

## JSON

```
// What can it store?

{
    "name": "fred", // strings
    "age": 22, // numbers
    "alergies": true, // boolean
    "likes": ["cats", "hugs"], // arrays
    "parents": { // objects
        "mom": "jane", "dad": "jim"
    }
}
```

## AJAX

## AJAX

**A**synchronous
**J**avaScript
**A**nd
**X**ML (Extensible Markup Language)

## JOE and JON AGREE THAT IT SHOULD BE…

**A**synchronous
**J**avaScript
**A**nd
**D**ata (Data)

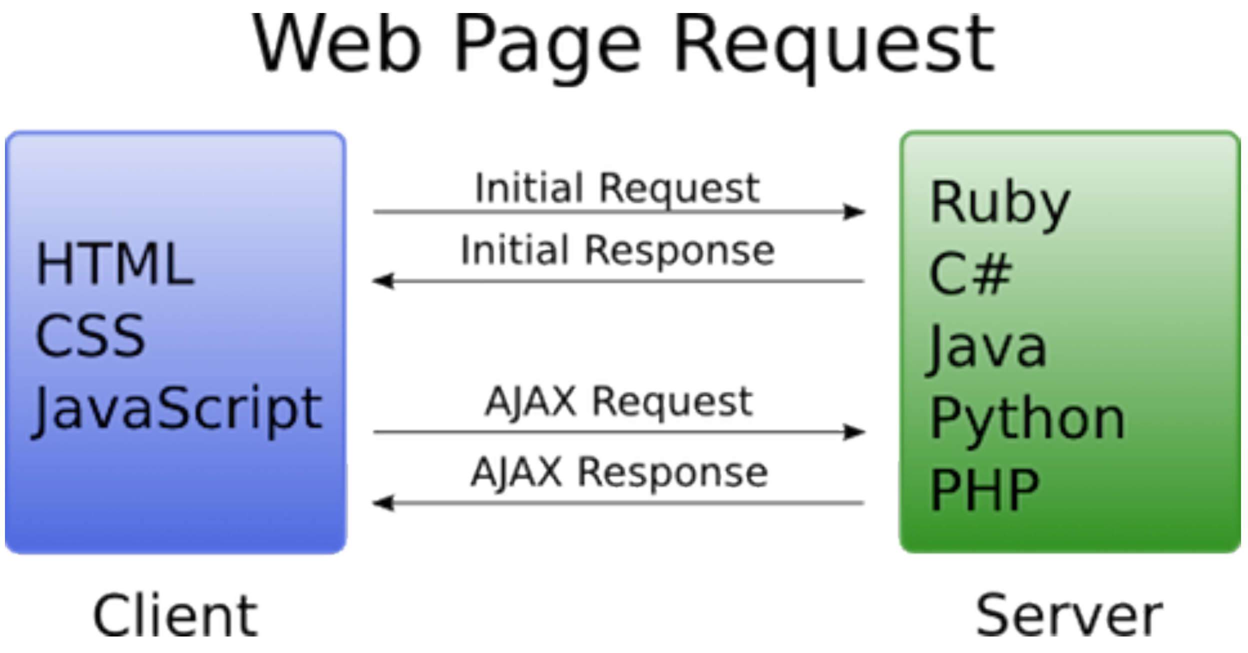
## Comparison of XML and JSON

```
<!– XML Example Code -->

<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd St.</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
</person>
```

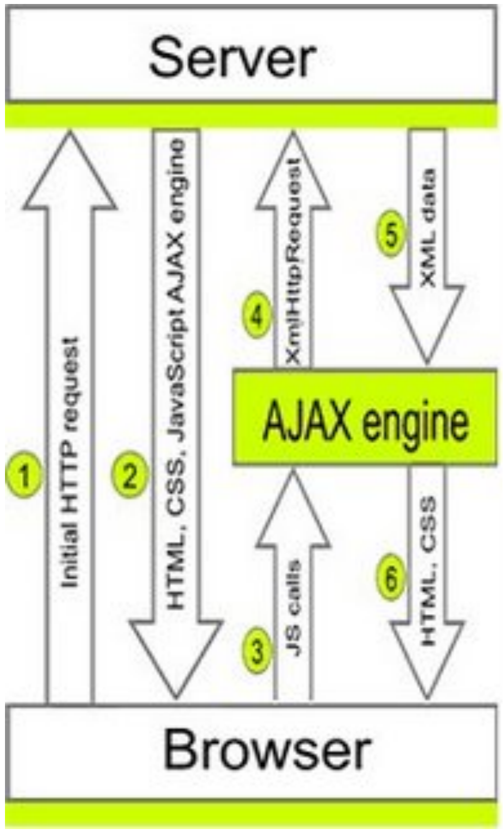
```
// JSON Example Code

{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd St.",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    }
}
```

## AJAX Describes a Process

## AJAX Request



## AJAX Request Breaking It Down



1. Initial request by the browser – the user requests the particular URL.

2. The complete page is rendered by the server (along with the JavaScript AJAX engine) and sent to the client (HTML, CSS, JavaScript AJAX engine).

3. All subsequent requests to the server are initiated as function calls to the JavaScript engine.

4. The JavaScript engine then makes an XmlHttpRequest to the server.

5. The server processes the request and sends a response in XML format to the client (XML document). It contains the data only of the page elements that need to be changed. In most cases this data comprises just a fraction of the total page markup.

6. The AJAX engine processes the server response, updates the relevant page content or performs another operation with the new data received from the server. (HTML + CSS)

## jQuery - AJAX Low Level Interface

```
$.ajax();               //Perform an asynchronous HTTP (Ajax) request.

$.ajaxPrefilter();      //Handle custom Ajax options or modify existing options before
                        each request is sent and before they are processed by $.ajax().

$.ajaxSetup();          //Set default values for future Ajax requests. Its use is not
                        recommended.

$.ajaxTransport();      //Creates an object that handles the actual transmission of
                        Ajax data.
```

## jQuery - AJAX Shorthand Methods

```
$.get();            //Load data from the server using a HTTP GET request.

$.getJSON();        //Load JSON-encoded data from the server using a GET HTTP request.

$.getScript();      //Load a JavaScript file from the server using a GET HTTP request,
                    then execute it.

$.post();           //Load data from the server using a HTTP POST request.

.load();            //Load data from the server and place the returned HTML into the
                    matched element.
```

## jQuery – Global AJAX Event Handlers

```
.ajaxComplete();   //Register a handler to be called when Ajax requests complete. This
                   is an AjaxEvent.

.ajaxError();      //Register a handler to be called when Ajax requests complete with
                   an error. This is an Ajax Event.

.ajaxSend();       //Attach a function to be executed before an Ajax request is sent.
                   This is an Ajax Event.

.ajaxStart();      //Register a handler to be called when the first Ajax request
                   begins. This is an Ajax Event.

.ajaxStop();       //Register a handler to be called when all Ajax requests have
                   completed. This is an Ajax Event.

.ajaxSuccess();    //Attach a function to be executed whenever an Ajax request
                   completes successfully. This is an Ajax Event.
```

## jQuery –AJAX Helper Functions

```
$.param();            //Create a serialized representation of an array or object,
                      suitable for use in a URL query string or Ajax request.

.serialize()          //Encode a set of form elements as a string for submission.

.serializeArray();    //Encode a set of form elements as an array of names and values.
```