JSON

Class 27

Data Formats

- our previous AJAX examples used
  - plain text (roses.txt)
  - text formatted as HTML (`get_words.php`)
- to transfer data from server to browser

```
foreach ($word_list as $array) : ?>
  <?php if (preg_match("/^$search.*/", $array[0])) : ?>
    <?php $wordpart = $array[0] . '_' . $array[1]; ?>
    <div>
      <input type="radio" value="<?= $wordpart ?>"
             name="wordradio" id="<?= $wordpart ?>" />
    </div>
    <div>
      <label for="<?= $wordpart ?>" ><?= $array[0] ?></label>
    </div>
    <div><?= $array[1] ?></div>
  <?php endif; ?>
<?php endforeach;
```

Try It

you can try it yourself: `https://borax.truman.edu/315/c26/get_words.php?search=cha`
(note this is get_words.php from the previous class)

```
                    <div>
      <input type="radio" value="chaff_noun"
             name="wordradio" id="chaff_noun" />
    </div>
    <div>
      <label for="chaff_noun" >chaff</label>
    </div>
    <div>noun</div>
                    <div>
      <input type="radio" value="chary_adjective"
             name="wordradio" id="chary_adjective" />
    </div>
    <div>
      <label for="chary_adjective" >chary</label>
    </div>
    <div>adjective</div>
```

this pre-formatted HTML code is plopped directly into the grid

Data Formats

- this works but is brittle
- PHP on the server and JS on the browser are tightly coupled
- the server source must produce data formatted exactly for HTML
- including CSS class names, ID's, etc.
- but most outside data sources are in different formats

- we want formats that are
  - HTML- and CSS-independent
  - interoperable
  - standards-compliant

Data Formats

- two formats are heavily used in web programming

  XML eXtensible Markup Language

  JSON JavaScript Object Notation

XML

- XML is a metalanguage
- XML is a language specification
- XML is a set of rules for making a language
- XML allows you to create a new language on the fly
- an XML language is used for data (content), not behavior
- HTML5 is an XML language
- many systems use an XML language for data storage
  - Microsoft Office (Office Open XML)
  - LibreOffice (OpenDocument)
  - Apple iWork
  - gnumeric spreadsheet

example: Gettysburg address in LibreOffice

XML Pros and Cons

- pros
  - very widely deployed
  - DTD allows for rules and syntax checking, robust
  - each XML comes with a DOM, so is searchable and easily manipulated
- cons
  - bulky
  - tedious to parse, requires a DOM
  - overkill for small amounts of data

JSON

- a competing data exchange format
- lightweight
- uses JavaScript syntax, so not another language to learn
- fully parsable without its own separate DOM

JS Objects

- recall JS objects that are created on the fly

```
const person =
{
  name: "Fred",
  address: "123 Rocky Path",
  friends: ["Barney", "Betty"],
  greet_wife: function()
  {
    return this.name + " loves Wilma";
  }
}

person.address;  // yields 123 Rocky Path
person.age = 50000;  // creates new field on the fly
person.greet_wife(); // returns Fred loves Wilma
```

JSON Data Format

- JSON is a specification for data format
- JSON is based on JS object syntax, but
  - can't have functions
  - attributes must in quotes
- here is a bit of data in JSON format

```
{
  "name": "Fred",
  "address": "123 Rocky Path",
  "friends": ["Barney", "Betty"]
}
```

JSON for Us

- we will not build JSON manually
- we will use JSON to communicate between server and client
- our typical case is that we have an associative array to pass

- for example, on the server, we have a PHP array
- we need to give the array to JS on the client
- we will use a couple of simple library routines to accomplish this via JSON

JSON Functions

- PHP:
  - turn a PHP array into a JSON-encoded string
    ```
    $json = json_encode($my_array);
    ```
  - turn a JSON-encoded string into a PHP array
    ```
    $my_array = json_decode($json);
    ```
- JS:
  - turn a JS array into a JSON-encoded string
    ```
    const json = JSON.stringify(my_array);
    ```
  - turn a JSON-encoded string into a JS array
    ```
    const my_array = JSON.parse(json);
    ```

- each of these assumes the thing being encoded is in fact an array

Template Literal

- earlier I told you that JS does not have string interpolation
- that's sort of true
- JS has a thing called a template literal

- a template literal is enclosed in backticks ` characters
- a template literal can contain placeholders like this `${expression}`
- a template literal can contain embedded newlines

String Interpolation

- a template literal is passed to a function
- if no function is specified, the default function simply converts the template into a string, interpolating any enclosed expressions

```
const me = "Bob";
const you = "Ann";

console.log(`Hello ${you}, I'm ${me}`);
```

- so a template literal works kind of like string interpolation
- if you know Python, it's more like the string .format method: `print("Hello {}, I'm {}\n".format(you, me))`

Example

selectword.html (different from previous class)

Sending Data

- we typically use GET for moving data from server to client
- we typically use POST for moving data from client to server
- just a couple of things are different for POST

```
xhr.open("POST", url);
xhr.setRequestHeader
  ("Content-Type",
   "application/x-www-form-urlencoded;charset=utf-8");
xhr.send(json_string);
```

- setRequestHeader must come after open() and before send()
- the argument of send() must be a string with both the data and a key for the $_POST array:
  `json_string = `key=${JSON.stringify(data)}`;`

Receiving Data

- how the PHP acquires the data varies depending on the original format in JS
- flavor 1: send a simple array

The JS:

```
const data_array = ["apple", "banana", "cabbage", "dates"];
const json_string = `payload=${JSON.stringify(data_array)}`;
  ...
xhr.send(json_string);
```

The PHP:

```
if (isset($_POST) && isset($_POST['payload']))
{
  $json_data = json_decode($_POST['payload']);
  foreach ($json_data as $key => $value)
```

`[0] => apple, [1] => banana, etc`

Receiving Data

- flavor 2: send a JSON object

The JS:

```
const data_object =
{
  "name": "Fred Flintstone",
  "address": "123 Rocky Way",
  "city": "Bedrock"
};
const json_string = `payload=${JSON.stringify(data_object)}`;
  ...
xhr.send(json_string);
```

The PHP:

```
if (isset($_POST) && isset($_POST['payload']))
{
  $json_data = json_decode($_POST['payload']);
  foreach ($json_data as $key => $value)
```

['name'] => Fred Flintstone, ['address'] => 123 Rocky Way,