

JSON Auto-Suggest Search v.5

How to Use this Script

You'll need:

- an input field with an id
- another element like a <div> with an id where your search results will be added.
- JSON object with the data to search in. Read more about JSON objects below.

5 Minute Implementation

1. Upload folder to your server.
2. Modify contents of /js/data_jsonp.json inputting your data, formatted as a JSON object or substitute this JSON object with your own.
3. If the structure of your JSON object is different than the example, update /js/index.js to search your JSON structure. See below for more info and examples.

JSON Formatted Objects

JSON is short for JavaScript Object Notation, and is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a logical manner.

Here's a simple example of how data about a famous person might be stored in JSON format:

```
var gClooney = {  
    "age" : "55",  
    "hometown" : "Lexington, Kentucky",  
    "gender" : "male"  
};
```

This creates an object that we access using the variable gClooney. By enclosing the variable's value in curly braces, we're indicating that the value is an object. Inside the object, we can declare any number of properties using a "name": "value" pairing, separated by commas. To access the information stored in gClooney, we can simply refer to the name of the property we need. For instance, we could use the following snippets:

```
document.write('George Clooney is ' + gClooney.age); // Output: George Clooney is 55  
document.write('Jason is a ' + gClooney.gender); // Output: George Clooney is a male
```

Storing JSON Data in Arrays

A slightly more complicated example involves storing two people in one variable. To do this, we enclose multiple objects in square brackets, which signifies an array. For instance, if I needed to include information about myself and my brother in one variable, I might use the following:

```
var superstars = [{
  "name" : "George Clooney",
  "age" : "55",
  "gender" : "male"
},
{
  "name" : "Britney Spears",
  "age" : "36",
  "gender" : "female"
}];
```

To access this information, we need to access the array index of the person we wish to access. For example, we would use the following snippet to access info stored in superstars:

```
document.write(superstars[1].name); // Output: Britney Spears
document.write(superstars[0].age); // Output: 55
```

Another way to store multiple people in our variable would be to nest objects. To do this, we would create something similar to the following:

```
var superstars = {
  "actor" : {
    "name" : "George Clooney",
    "age" : "55",
    "gender" : "male"
  },
  "singer" : {
    "name" : "Britney Spears",
    "age" : "36",
    "gender" : "female"
  }
}
```

Accessing information in nested objects is a little easier to understand; to access information in the object, we would use the following snippet:

```
document.write(superstars.singer.name); // Output: Britney Spears
document.write(superstars.singer.age); // Output: 36
document.write(superstars.actor.gender); // Output: male
```

Nested JSON and arrays can be combined as needed to store as much data as necessary.

Why is JSON important?

With the rise of AJAX-powered sites, it's becoming more and more important for sites to be able to load data quickly and asynchronously, or in the background without delaying page rendering.

Because of the popularity and ease of social media, many sites rely on the content provided by sites such as Twitter, Flickr, and others. These sites provide RSS feeds, which are easy to import and use on the server-side, but if we try to load them with AJAX, we run into a wall: we can only load an RSS feed if we're requesting it from the same domain it's hosted on.

JSON allows us to overcome the cross-domain issue because we can use a method called JSONP that uses a callback function to send the JSON data back to our domain. It's this capability that makes JSON so incredibly useful, as it opens up a lot of doors that were previously difficult to work around.

To check if your JSON data is formatted correctly, look for a JSON validator online. We like <https://jsonformatter.curiousconcept.com/>

How to convert your data to JSON format

There are several ways to do this in javascript, but perhaps the easiest is to use `JSON.stringify`, which converts a JavaScript value to a JSON string:

```
JSON.stringify(value[, replacer[, space]])
```

For example, start with any array. Here is a simple example, but you can use arrays with multiple levels as well:

```
var superstar = {"name" : "George Clooney"};
```

Then use `stringify` to format that array into a JSON object:

```
JSON.stringify(superstar); // output: '{"name":"George Clooney"}'
```

Read the official documentation here: <https://mzl.la/1fms8sL>

Generating JSON with PHP

Start with any array in PHP:

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
?>
```

Use the function `json_encode` to format it into a JSON object, like this:

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
echo json_encode($arr);
?>
```

This will produce the following output, JSON object:

```
{ "a":1,"b":2,"c":3,"d":4,"e":5 }
```

How to make this script work with your MySQL database

To quickly make this script work reading data from your MySQL table:

1. Upload the file `data_json.php` to your server and change the path in the js file `/js/search_light.js` (or `_dark.js` or `_grid.js` depending on which example you're using), line 61, to point to this new php file.
2. If you don't have a database yet, create a MySQL database and assign user privileges. Import the SQL file to create a table with demo data.
3. Open the `data_json.php` you just uploaded and insert your database connection information (lines 2 - 5).

The script will work, reading info from the demo database. When you're ready, you can change the database table query and field names in `data_jsonp.php` (lines 15 & 21-24) to match your own table and field names.

Dealing with symbols in JSON properties

Sometimes you'll find a symbol in the JSON property, as in the following example:

```
{
  "page":[
    {
      "@attributes":{
```

```

        "id": "1"
      },
      "keywords": "Car Motors\n\nBrakes, mufflers"
    },
    {
      "@attributes": {
        "id": "2"
      },
      "keywords": "Air Conditioning\nCoolant, fans"
    }
  ]
}

```

Since symbols are not allowed in javascript variables, we can't access this son element in the usual way. For example, if the property was "attributes" instead of "@attributes", we could access it by using

```
key.attributes
```

but

```
key.@attributes
```

will return an error.

When you have symbols in your JSON properties, you should access the item using **square brackets**, like this:

```
myjson["@propertyname"]
```

```
myjsonobject[index]["@attributes"].id
```

So, to access "id" in the example above, we could do this:

```
var data = json.page
```

```

function useParsedData(data) {
  data.forEach(function(key, index, val) {
    console.log( data[index]['@attributes'].id );
  })
}

```

Javascript Regular Expressions

This script searches for each keyword inserted in the input field, but you can easily alter the regular expression to meet your search needs.

For example, if you want the search to return exclusively items with the phrase "in front" (inputted words, in order they were inputted), and *not* all items that have either "in" or "front", you can change the searchField event listener to the following:

```
searchField.addEventListener("keyup", function() {
```

```
var searchInput = searchField.value;  
regex = new RegExp(searchInput, "i");  
useParsedData(jsonData);  
});
```

As of release 0.4, the script searches all JSON items that have either/any of the user-inserted keywords. For example, if you search for “blue yellow” it will return:

- all items that have blue
- all items that have yellow
- all items that have the words blue *and* yellow in any order

If you search for “blue yellow red”, it will return

- all items that have blue
- all items that have yellow
- all items that have red
- all items that have the words blue *and* yellow *and* red in any order

For more information on regular expressions, we recommend:

http://www.w3schools.com/jsref/jsref_obj_regexp.asp