



## JSON

JSON stands for **JavaScript Object Notation**.

JSON is a **plain text format** for storing and transporting data.

JSON is similar to the syntax for creating JavaScript objects.

JSON is used to **send, receive** and **store data**.

## JSON Data - A Name and a Value

JSON data is written as name/value pairs (aka key/value pairs).

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

### Example

```
"name": "John"
```

The JSON format is almost identical to JavaScript objects.

In JSON, *keys* must be strings, written with double quotes:

## JSON

```
{"name": "John"}
```

In JavaScript, keys can be strings, numbers, or identifier names:

## JavaScript

```
{name: "John"}
```

# JSON and JavaScript

The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this, a JavaScript program can easily convert JSON data into native JavaScript objects.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

`JSON.parse()`

JavaScript also has a built in function for converting an object into a JSON string:

`JSON.stringify()`

You can receive pure text from a server and use it as a JavaScript object.

You can send a JavaScript object to a server in pure text format.

You can work with data as JavaScript objects, with no complicated parsing and translations.

# JSON Data Types

## Valid Data Types

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

# JSON Strings

Strings in JSON must be written in double quotes.

## Example

```
{"name": "John"}
```

# JSON Numbers

Numbers in JSON must be an integer or a floating point.

## Example

```
{"age": 30}
```

# JSON Objects

Values in JSON can be objects.

## Example

```
{  
  "employee": {"name": "John", "age": 30, "city": "New York"}  
}
```

# JSON Arrays

Values in JSON can be arrays.

## Example

```
{  
  "employees": ["John", "Anna", "Peter"]  
}
```

# JSON Booleans

Values in JSON can be true/false.

## Example

```
{"sale":true}
```

# JSON null

Values in JSON can be null.

## Example

```
{"middlename":null}
```

## Example:1

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript JSON</h1>
<h2>Creating an Object from JSON</h2>

<p id="demo"></p>

<script>
const txt = '{"name":"John", "age":30, "city":"New York"}'
const myObj = JSON.parse(txt);

document.getElementById("demo").innerHTML = myObj.name + ", " +
myObj.age;
</script>

</body>
</html>
```

# JavaScript JSON

## Creating an Object from JSON

John, 30

## Example:2

```
<!DOCTYPE html>
<html>
<body>

<h2>Parsing a JSON Array.</h2>
<p>Data written as an JSON array will be parsed into a JavaScript array.
</p>
<p id="demo"></p>

<script>
const text = '[ "Ford", "BMW", "Audi", "Fiat" ]';
const myArr = JSON.parse(text);
document.getElementById("demo").innerHTML = myArr[0];
</script>

</body>
</html>
```

## Parsing a JSON Array.

Data written as an JSON array will be parsed into a JavaScript array.

Ford

Functions are not allowed in JSON.

If you need to include a function, write it as a string.

You can convert it back into a function later.

## Example :3

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a string into a function.</h2>
<p id="demo"></p>

<script>
const text = '{"name":"John", "age":"function() {return 30;}",
"city":"New York"}';
const obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();
</script>

</body>
</html>
```

**Convert a string into a function.**

John, 30

# JSON.stringify()

A common use of JSON is to exchange data to/from a web server.

When sending data to a web server, the data has to be a string.

You can convert any JavaScript datatype into a string with `JSON.stringify()`.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript JSON</h1>
<h2>Create a JSON string from ant object.</h2>
<p id="demo"></p>

<script>
const myObj = {name: "John", age: 30, city: "New York"};
const myJSON = JSON.stringify(myObj);

document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

## JavaScript JSON

Create a JSON string from ant object.

```
{"name":"John","age":30,"city":"New York"}
```

## Example :4

```
<!DOCTYPE html>
<html>
<body>

<h2>Create a JSON string from a JavaScript array.</h2>
<p id="demo"></p>

<script>
const arr = ["John", "Peter", "Sally", "Jane"];
const myJSON = JSON.stringify(arr);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

**Create a JSON string from a JavaScript array.**

["John","Peter","Sally","Jane"]

## Example :6

```
<!DOCTYPE html>
<html>
<body>

<h2>Store and retrieve data from local storage.</h2>
<p id="demo"></p>

<script>
// Storing data:
const myObj = { name: "John", age: 31, city: "New York" };
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>

</body>
</html>
```

**Store and retrieve data from local storage.**

John

# Looping an Object

## Example :7

```
<!DOCTYPE html>
<html>
<body>

<h2>Looping Object Properties</h2>
<p id="demo"></p>

<script>
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += x + ", ";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

## Looping Object Properties

name, age, car,



## Example :8

```
<!DOCTYPE html>
<html>
<body>

<h2>Looping JavaScript Object Values</h2>
<p id="demo"></p>

<script>
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += myObj[x] + ", ";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

## Looping JavaScript Object Values

John, 30, null,

# Arrays in Objects

## Example :9

```
<!DOCTYPE html>
<html>
<body>

<h2>Access Array Values</h2>
<p id="demo"></p>

<script>
const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW",
"Fiat"]}';
const myObj = JSON.parse(myJSON);

document.getElementById("demo").innerHTML = myObj.cars[0];
</script>

</body>
</html>
```

## Access Array Values

Ford

## Example :10

```
<!DOCTYPE html>
<html>
<body>

<h2>Looping an Array</h2>
<p id="demo"></p>

<script>
const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW",
"Fiat"]}';
const myObj = JSON.parse(myJSON);

let text = "";
for (let i in myObj.cars) {
  text += myObj.cars[i] + ", ";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

## Looping an Array

Ford, BMW, Fiat,

# JSON Server

## Sending Data

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a JavaScript object into a JSON string, and send it to the
server.</h2>
```

```
<script>
const myObj = { name: "John", age: 31, city: "New York" };
const myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
</script>
```

```
</body>
</html>
```

demo\_json.php:

John from New York is 31



# Receiving Data

If you receive data in JSON format, you can easily convert it into a JavaScript object

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a JSON string into a JavaScript object.</h2>

<p id="demo"></p>

<script>
const myJSON = '{"name":"John", "age":31, "city":"New York"}';
const myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

**Convert a JSON string into a JavaScript object.**

John

# JSON From a Server

- You can request JSON from the server by using an AJAX request
- As long as the response from the server is written in JSON format, you can parse the string into a JavaScript object.



```
<!DOCTYPE html>
<html>
<body>

<h2>Fetch a JSON file with XMLHttpRequest</h2>
<p id="demo"></p>
```

```
<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "json_demo.txt");
xmlhttp.send();
</script>
```

## Fetch a JSON file with XMLHttpRequest

John

```
</body>
</html>
```

# Array as JSON

When using the `JSON.parse()` on JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

```
<!DOCTYPE html>
<html>
<body>

<h2>Fetch a JSON file with XMLHttpRequest</h2>
<p>Content written as an JSON array will be converted into a JavaScript
array.</p>
<p id="demo"></p>

<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myArr = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myArr[0];
}
xmlhttp.open("GET", "json_demo_array.txt", true);
xmlhttp.send();
</script>

</body>
</html>
```

## Fetch a JSON file with XMLHttpRequest

Content written as an JSON array will be converted into a JavaScript array.

Ford

## JSON PHP

- PHP has some built-in functions to handle JSON.
- Objects in PHP can be converted into JSON by using the PHP function json\_encode().

```
<?php
$myObj = new stdClass();
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";
                                         {"name":"John","age":30,"city":"New York"}
$myJSON = json_encode($myObj);

echo $myJSON;
?>
```

## The Client JavaScript

Here is a JavaScript on the client, using an AJAX call to request the PHP file from the previous example.

```
<!DOCTYPE html>
<html>
<body>

<h2>Get JSON Data from a PHP Server</h2>
<p id="demo"></p>

<script>
const xmlhttp = new XMLHttpRequest();

xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "demo_file.php");
xmlhttp.send();
</script>

</body>
</html>
```

## Get JSON Data from a PHP Server

John

## PHP Array

Arrays in PHP will also be converted into JSON when using the PHP function json\_encode()

```
<?php
$myArr = array("John", "Mary", "Peter", "Sally");
$myJSON = json_encode($myArr);                                ["John","Mary","Peter","Sally"]
echo $myJSON;
?>
```

Use JSON.parse() to convert the result into a JavaScript array

```
<!DOCTYPE html>
<html>
<body>

<h2>Get JSON Data from a PHP Server</h2>
<p>Convert the data into a JavaScript array:</p>
<p id="demo"></p>

<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
    const myObj = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myObj[2];
}
xmlhttp.open("GET", "demo_file_array.php");
xmlhttp.send();
</script>

</body>
</html>
```

## Get JSON Data from a PHP Server

Convert the data into a JavaScript array:

Peter

# jQuery

jQuery is a **JavaScript library** (a collection of pre-written code) created to make web development easier and faster.

It mainly helps with:

- Selecting and changing HTML elements
- Handling events (click, hover, etc.)
- Creating animations/effects
- Sending/receiving data from servers (AJAX)
- Making code **shorter, cleaner, and cross-browser compatible**

```
// Vanilla JS
document.getElementById("title").style.color = "red";

// jQuery
$("#title").css("color", "red");
```

```
$("#btn").click(function() {
    alert("Button clicked!");
});
```

```
$(".box").css("color", "red");
```

```
$.get("data.json", function(data) {
    console.log(data);
});
```

# jQuery Syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **`$(selector).action()`**

- A \$ sign to define/access jQuery
- A *(selector)* to "query (or find)" HTML elements
- A jQuery *action()* to be performed on the element(s)

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all `<p>` elements.

`$(".test").hide()` - hides all elements with `class="test"`.

`$("#test").hide()` - hides the element with `id="test"`.

# **jQuery Effects**

## **1. jQuery Hide/Show** □ Used for simple visibility toggling.

`hide()` → makes an element invisible.

`show()` → displays the hidden element.

## **2. jQuery Fade** □ Used for smooth transitions

`fadeIn()` → slowly makes element visible.

`fadeOut()` → slowly hides element.

`fadeToggle()` → toggles between `fadeIn` & `fadeOut`.

## **3. jQuery Slide** □ Great for menus, dropdowns, accordions.

`slideDown()` → slides content down (makes visible).

`slideUp()` → slides content up (hides).

`slideToggle()` → toggles between slide up/down.

#### **4. jQuery Animate** □ Used for custom animations.

animate() → lets you change CSS properties (position, size, opacity) smoothly. You can specify duration and multiple CSS styles.

#### **5. jQuery stop()** □ Prevents queue buildup when you click multiple times.

stop() → stops the current animation or effect immediately.

#### **6. jQuery Callback** □ Ensures tasks run in order.

A callback function runs after an effect is completed.

#### **7. jQuery Chaining** □ Saves code and ensures smooth execution.

Chaining = run multiple jQuery methods on the same element in one line.

# jQuery HTML

## **1. jQuery Get**

.text(), .html(), .val() can get values from elements.

## **2. jQuery Set**

Same methods .text(), .html(), .val() can set values when argument is passed.

## **3. jQuery Add**

.append() adds content inside element, .prepend() adds at the beginning, .after() & .before() insert outside.

## **4. jQuery Remove**

.remove() deletes the element entirely, .empty() clears only its content.

## **5. jQuery CSS Classes**

.addClass(), .removeClass(), .toggleClass() manage CSS classes dynamically.

## **6. jQuery css()**

.css() gets or sets inline CSS styles.