Jason? No, JSON!

Jason is an ancient Greek mythological hero who went traipsing about the known world looking for "the golden fleece".

JSON is an open-standard file format that "uses human-readable text to transmit objects consisting of key-values pairs and array data types."

We're going to ignore Jasonand focus solely on JSON for this reading so that you can, by the end of it, know what JSON is and how to work with it.

JSON is a format!

This is the most important thing that you can get when reading this article. In the same way that HTML is a format for hypertext documents, or DOCX is a format for Microsoft Word documents, JSON is just a format for data. It's just text. It doesn't "run" like JavaScript does. It is just text that contains data that both machines and humans can understand. If you ever hear someone say "a JSON object", then you can rest assured that phrase doesn't make any sense whatsoever.

JSON is just a string. It's just text.

That's so important, here it is, again, but in a fancy quote box.

JSON is just a string. It's just text.

Why all the confusion?

The problem is, JSON *looks* a lot like JavaScript syntax. Heck, it's even named**JavaScript Object Notation**. That's likely because the guy who invented it,Douglas Crockford, is an avid JavaScripter. He's the author of JavaScript: The Good Partsand was the lead JavaScript Architect at Yahoo! back when Yahoo! was a real company.

At that time, like in the late 1990s and early 2000s, there were a whole bunch of competing formats for how computers would send data between one another. The big contender at the time is a format called XML, or the *eXtensible Markup Language*. It looks a lot like HTML, but has far stricter rules than HTML. Douglas didn't like XML because it took a lot of bytes to send the data (and this was a pre-broadband/pre-3G world). Worse, XML is not a friendly format to read if you're human. So, he set out to come up with a new format based on the way JavaScript literals work.

"Remind me about JavaScript literals..."

Just to refresh your memory, a *literal*in JavaScript is a *value that you literally just type in*. If you type `7`into a JavaScript file, when it runs, the JavaScript interpreter will see that character `7` and say to itself, "Hey self, the programmer literally typed the number seven so that must mean they want the value 7."

Here's a table of some literals that you may type into a program.

What you want to type	The JavaScript literal
The value that means "true"	`true`
The number of rows in this table	`6`

What you want to type	The JavaScript literal
A bad approximation of π	`3.14`
An array that contains some US state names	`["Ohio", "Iowa"]`
An object that represents Roberta	`{ person: true, name: "Roberta" }`

Back to Douglas Crockford, inventor of JSON. Douglas thought to himself, *why can't I create a format that has that simplicity so that I can write programs that can send data to each other in that format?*Turns out, he could, and he did.

Boolean, numeric, and null values

The following table shows you what the a JavaScript literal is in the JSON format. Notice that *everything*in the JSON column is actually a string!

JavaScript literal value	JSON representation in a string
`true`	`"true"`
`false`	`"false"`
`12.34`	`"12.34"`
`null`	`"null"`

String literals in JSON

Say you have the following string in JavaScript.

```
'this is "text"'
```

When that gets converted into the JSON format, you will see this:

```
"this is \"text\""
```

First, it's important to notice one thing: JSON always uses double quotes for strings. Yep, that's worth repeating.

JSON always uses double-quotes to mark strings.

Notice also that the quotation marks (") are "escaped". When you write a string surrounded by quotation-marks like "escaped", everything's fine. But, what happens when your string needs to include a quotation mark?

```
// This is a bad string with quotes in it
"Bob said, "Well, this is interesting.""
```

Whatever computer is looking at that string gets really confused because once it reads that first quotation mark it's looking for another quotation mark to show where the string ends. For computers, the above code looks like this to them.

```
"Bob said, "               // That's a good string
Well, this is interesting  // What is THIS JUNK????
""                         // That's a good string
```

You need a way to indicate that the quotation marks around the phrase that Bob says should belong *in*the string, not as a way to show where the string

starts or stops. The way that language designers originally addressed this was by saying

If your quotation mark delimited string has a quotation mark in it, put a backslash before the interior quotation mark.

Following that rule, you would correctly write the previous string like this.

```
"Bob said, \"Well, this is interesting.\""
```

Check out all of the so-called JavaScript string escape sequencesover on MDN.

What happens if you had text that spanned more than one line? JSON only allows strings to be on one line, just like old JavaScript did. Let's say you just wrote an American sentence that you want to submit to a contest.

```
She woke him up with
her Ramones ringtone "I Want
to be Sedated"
```

(from American Sentences by Paul E. Nelson)

If you want to format that in a string in JSON format, you have to escape the quotation marks *and* the new lines! The above would look like this:

```
She woke him up with\nher Ramones ringtone \"I Want\nto be Sedated\"
```

The new lines are replaced with "\n".

Array values

The way that JSON represents an array value is using the same literal notation as JavaScript, namely, the square brackets `[]`. With that in mind, can you answer the following question before continuing?

What is the JSON representation of an array containing the numbers one, two, and three?

Well, in JavaScript, you would type `[1, 2, 3]`.

If you were going to type the corresponding JSON-formatted string that contains the representation of the same array, you would type `"[1, 2, 3]"`. Yep, pretty much the same!

Object values

Earlier, you saw that example of an object that represents Roberta as

```
{ person: true, name: "Roberta" }
```

The main difference between objects in JavaScript and JSON is that the keys in JSON *must* be surrounded in quotation marks. That means the above, in a JSON formatted string, would be:

```
"{ \"person\": true, \"name\": \"Roberta\" }"
```

Some terminology

When you have some data and you want to turn it into a string (or some other kind of value like "binary") so your program can send it to another computer,

that is the process of **serialization**.

When you take some text (or something another computer has sent to your program) and turn it into data, that is the process of **deserialization**.

Using the built-in JSON object

In modern JavaScript interpreters, there is a `JSON` object that has two methods on it that allows you to convert JSON-formatted strings into JavaScript objects and JavaScript object into JSON-formatted strings. They are:

- `JSON.stringify(value)` will turn the value passed into it into a string.
- `JSON.parse(str)` will turn a JSON-formatted string into a JavaScript object.

So, it shouldn't come as much of a surprise how the following works.

```
const array = [1, 'hello, "world"', 3.14, { id: 17 }];
console.log(JSON.stringify(array));
// prints [1, "hello, \"world\"", 3.14, {"id":17}]
```

It shouldn't surprise you that it works in the opposite direction, too.

```
const str = '[1,"hello, \\"world\\"",3.14,{"id":17}]';
console.log(JSON.parse(str));
// prints an array with the following entries:
//   0: 1
//   1: "hello, \"world\""
//   2: 3.14
//   3: { id: 17 }
```

You may ask yourself, "What's up with that double backslash thing going on in the JSON representation?". It has to do with that escaping thing. When JavaScript reads the string the first time to turn it into a `String` object in memory, it will escape the backslashes. Then, when `JSON.parse` reads it, it will still need backslashes in the string. This is all really confusing, escaped strings and double backslashes. There's an easy solution for that.

You will almost never write raw JSON

Yep. But, you do need to be able to recognize it and read it. What you'll likely end up doing in your coding is creating values and using `JSON.stringify` to create JSON-formatted strings that represent those values. Or, you'll end up calling a data service which will return JSON-formatted content to your code which you will then use `JSON.parse` on to convert the string into a JavaScript object.

Brain teaser

Now that you know JSON is a format for data and is just text, what will the following print?

```
const a = [1, 2, 3, 4, 5];
console.log(a[0]);

const s = JSON.stringify(a);
console.log(s[0]);

const v = JSON.parse(s);
console.log(v[0]);
```

What you just learned

With some more practiced, of course, you will be able to do all of these really well. However, right now, you should be able to

1. Identify and generate valid JSON-formatted strings
2. Use `JSON.parse` to deserialize JSON-formatted strings
3. Use `JSON.stringify` to serialize JavaScript objects
4. Correctly identify the definition of "deserialize"
5. Correctly identify the definition of "serialize"