The **`JSON.parse()`** method parses a JSON string, constructing the JavaScript value or object described by the string. An optional **reviver** function can be provided to perform a transformation on the resulting object before it is returned.

Syntax

```
JSON.parse(text[, reviver])
```

Parameters

`text`
The string to parse as JSON. See the `JSON` object for a description of JSON syntax.

`reviver` | Optional
If a function, this prescribes how the value originally produced by parsing is transformed, before being returned.

Return value

The `Object`, `Array`, string, number, boolean, or null value corresponding to the given JSON *`text`*.

Exceptions

Throws a `SyntaxError` exception if the string to parse is not valid JSON.

Polyfill

```
// From https://github.com/douglascrockford/JSON-js/blob/master/json2.j
if (typeof JSON.parse !== "function") {
    var rx_one = /^[\],:{}\s]*$/;
    var rx_two = /\\(?:["\\/bfnrt]|u[0-9a-fA-F]{4})/g;
```

```
var rx_two = /\\(?:["\\\/bfnrt]|u[0-9a-fA-F]{4})/g;
var rx_three = /"[^"\\\n\r]*"|true|false|null|-?\d+(?:\.\d*)?(?:[eE
var rx_four = /(?:^|:|,)(?:\s*\[)+/g;
var rx_dangerous = /[\u0000\u00ad\u0600-\u0604\u070f\u17b4\u17b5\u2

JSON.parse = function(text, reviver) {

    // The parse method takes a text and an optional reviver functi
    // a JavaScript value if the text is a valid JSON text.

    var j;

    function walk(holder, key) {

        // The walk method is used to recursively walk the resultin
        // that modifications can be made.

        var k;
        var v;
        var value = holder[key];
        if (value && typeof value === "object") {
            for (k in value) {
                if (Object.prototype.hasOwnProperty.call(value, k))
                    v = walk(value, k);
                    if (v !== undefined) {
                        value[k] = v;
                    } else {
                        delete value[k];
                    }
                }
            }
        }
        return reviver.call(holder, key, value);
    }


    // Parsing happens in four stages. In the first stage, we repla
    // Unicode characters with escape sequences. JavaScript handles
    // incorrectly, either silently deleting them, or treating them

    text = String(text);
    rx_dangerous.lastIndex = 0;
```

```
rx_dangerous.lastIndex = 0;
if (rx_dangerous.test(text)) {
    text = text.replace(rx_dangerous, function(a) {
        return (

            "\\u" +
            ("0000" + a.charCodeAt(0).toString(16)).slice(-4)
        );
    });
}

// In the second stage, we run the text against regular express
// for non-JSON patterns. We are especially concerned with "()"
// because they can cause invocation, and "=" because it can ca
// But just to be safe, we want to reject all unexpected forms.

// We split the second stage into 4 regexp operations in order
// crippling inefficiencies in IE's and Safari's regexp engines
// replace the JSON backslash pairs with "@" (a non-JSON charac
// replace all simple value tokens with "]" characters. Third,
// open brackets that follow a colon or comma or that begin the
// we look to see that the remaining characters are only whites
// "," or ":" or "{" or "}". If that is so, then the text is sa

if (
    rx_one.test(
        text
        .replace(rx_two, "@")
        .replace(rx_three, "]")
        .replace(rx_four, "")
    )
) {

    // In the third stage we use the eval function to compile t
    // JavaScript structure. The "{" operator is subject to a s
    // in JavaScript: it can begin a block or an object literal
    // in parens to eliminate the ambiguity.

    j = eval("(" + text + ")");

    // In the optional fourth stage, we recursively walk the ne
    // each name/value pair to a reviver function for possible
```

```
                // each name/value pair to a reviver function for possible

                return (typeof reviver === "function") ?
                    walk({

                        "": j
                    }, "") :
                    j;
            }

            // If the text is not JSON parseable, then a SyntaxError is thr

            throw new SyntaxError("JSON.parse");
        };
    }
```

Examples

Using JSON.parse()

```
JSON.parse('{}');              // {}
JSON.parse('true');            // true
JSON.parse('"foo"');           // "foo"
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');            // null
```

Using the reviver parameter

If a *`reviver`* is specified, the value computed by parsing is *transformed* before being returned. Specifically, the computed value and all its properties (beginning with the most nested properties and proceeding to the original value itself) are individually run through the *`reviver`*. Then it is called, with the object containing the property being processed as `this`, and with the property name as a string, and the property value as arguments. If the *`reviver`* function returns `undefined` (or returns no value, for example, if execution falls off the end of the function), the property is deleted from the object. Otherwise, the property is redefined to be the return value.

If the *reviver* only transforms some values and not others, be certain to return all untransformed values as-is, otherwise, they will be deleted from the resulting object.

```
JSON.parse('{"p": 5}', (key, value) =>
  typeof value === 'number'
    ? value * 2 // return value * 2 for numbers
    : value     // return everything else unchanged
);

// { p: 10 }

JSON.parse('{"1": 1, "2": 2, "3": {"4": 4, "5": {"6": 6}}}', (key, valu
  console.log(key); // log the current property name, the last is "".
  return value;     // return the unchanged property value.
});

// 1
// 2
// 4
// 6
// 5
// 3
// ""
```

JSON.parse() does not allow trailing commas

```
// both will throw a SyntaxError
JSON.parse('[1, 2, 3, 4, ]');
JSON.parse('{"foo" : 1, }');
```

JSON.parse() does not allow single quotes

```
// will throw a SyntaxError
JSON.parse("{'foo': 1}");
```

Specifications

Specification
ECMAScript (ECMA-262) The definition of 'JSON.parse' in that specification.

Browser compatibility

Update compatibility data on GitHub

parse	
Chrome	3
Edge	12
Firefox	3.5
IE	8
Opera	10.5
Safari	4
WebView Android	≤37
Chrome Android	18
Firefox Android	4
Opera Android	11
Safari iOS	4
Samsung Internet Android	1.0
nodejs	0.1.100

What are we missing?

.. Full support