

# Q

**What are the best ways to transfer data?**

# Two main forms of (structured) data transfer

## XML

```
<employees>
  <employee>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
    <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName>
    <lastName>Jones</lastName>
  </employee>
</employees>
```

## JSON

```
{"employees": [
  { "firstName": "John",
    "lastName": "Doe" },
  { "firstName": "Anna",
    "lastName": "Smith" },
  { "firstName": "Peter",
    "lastName": "Jones" }
]}
```

# XML vs JSON

Both JSON and XML:

- are "self describing" (human readable)
- are hierarchical (values within values)
- can be parsed and used by lots of programming languages
- can be fetched with an XMLHttpRequest

For AJAX applications, JSON is faster and easier than XML:

## XML

Fetch an XML document  
Use the XML DOM to loop through the document  
Extract values and store them in variables

## JSON

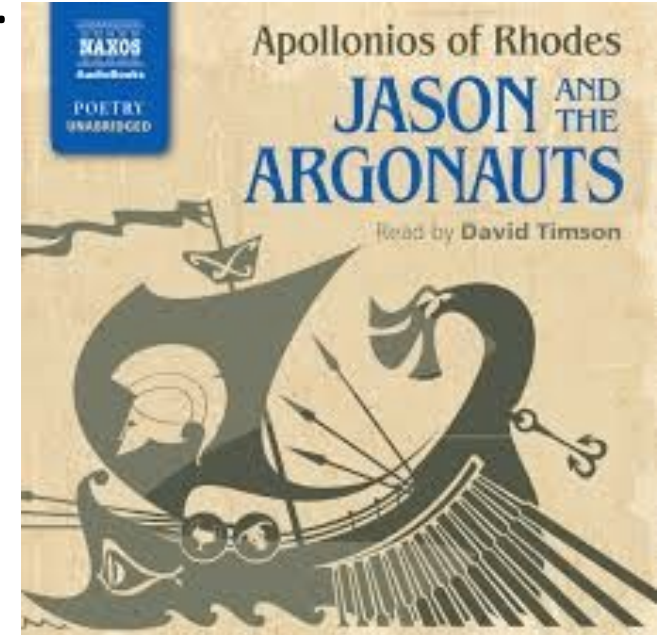
Fetch a JSON string  
JSON.Parse the JSON string



# JSON – JavaScript Object Notation

JSON is a language-independent data format.

```
{ "name": "Mario",  
  "surname": "Rossi",  
  "active": true,  
  "favoriteNumber": 42,  
  "birthday": {  
    "day": 1,  
    "month": 1,  
    "year": 2000  
  },  
  "languages": [ "it", "en" ]  
}
```



Datatypes:

int, float

Boolean

String

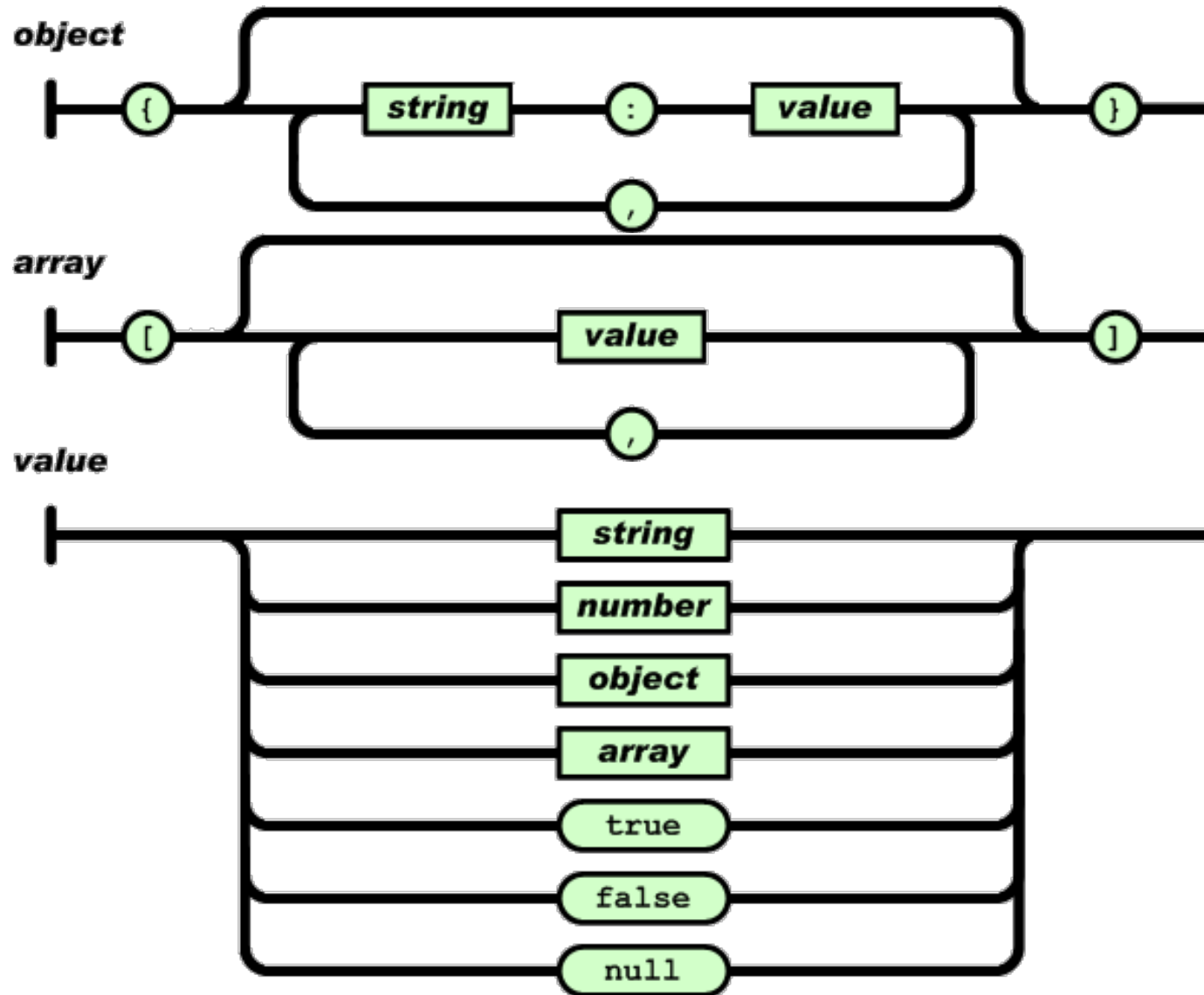
Arrays []

Associative Arrays {}

null



# JSON



# Parsing JSON in JavaScript

```
var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';  
var obj = JSON.parse(text);  
obj.birth = new Date(obj.birth);  
  
document.getElementById("demo").innerHTML = obj.name + ", " +  
obj.birth;
```



# Argo (Parsing JSON in Java)

```
{  
  "name": "Black Lace",  
  "sales": 110921,  
  "totalRoyalties": 10223.82,  
  "singles": [  
    "Superman", "Agadoo"  
  ]  
}
```

```
String secondSingle = new JdomParser().parse(jsonText)  
  .getStringValue("singles", 1);
```

**<http://argo.sourceforge.net/index.html>**



# AJAJ

```
var my_JSON_object;  
var url=" https://mdn.github.io/learning-  
area/javascript/oajs/json/superheroes.json"  
var xhttp = new XMLHttpRequest();  
xhttp.open("GET", url, true);  
xhttp.responseType = "json";  
xhttp.onreadystatechange = function () {  
    var done = 4, ok = 200;  
    if (this.readyState === done && this.status === ok)  
    {  
        my_JSON_object = this.response;  
    }  
};  
xhttp.send();}
```



# An example

<https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json>

```
{
  "squadName" : "Super Hero Squad",
  "homeTown" : "Metro City",
  "formed" : 2016,
  "secretBase" : "Super tower",
  "active" : true,
  "members" : [
    {
      "name" : "Molecule Man",
      "age" : 29,
      "secretIdentity" : "Dan Jukes",
      "powers" : [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name" : "Madame Uppercut",
      "age" : 39,
      "secretIdentity" : "Jane Wilson",
      "powers" : [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name" : "Eternal Flame",
      "age" : 1000000,
      "secretIdentity" : "Unknown",
      "powers" : [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

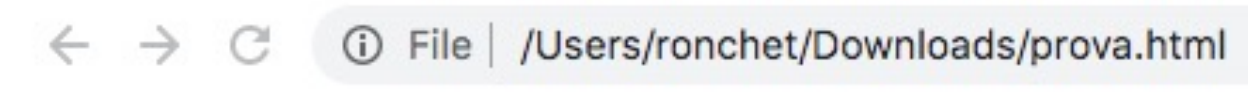
# The traps of asynchronous computing 1A

```
<script>
function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
        area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, true);
    xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
        var done = 4, ok = 200;
        if (this.readyState === done && this.status === ok){
            my_JSON_object = this.response;
        }
    };
    xhttp.send();
    return my_JSON_object;
}
</script>
```

# The traps of asynchronous computing 1B

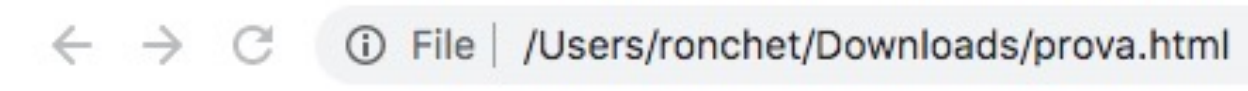
```
<!DOCTYPE html>
<head>
  <title>AJAJ Demo</title>
  <script>...</script>
</head>
<body>
  <form>
    <input type="BUTTON" onClick=
      'document.getElementById("myPar").innerHTML=getJson();' >
  </form>
  <p id="myPar">here the json will appear</p>
</body>
</html>
```

# The traps of asynchronous computing 1 out



Get it!

here the json will appear



Get it!

undefined

**Because the call is async!**

# The traps of asynchronous computing 2A

```
<script>
function getJson() {
  var my_JSON_object;
  var url="https://mdn.github.io/learning-
    area/javascript/oajs/json/superheroes.json"
  var xhttp = new XMLHttpRequest();
  xhttp.open("GET", url, false);
  xhttp.responseType = "json";
  xhttp.onreadystatechange = function () {
    var done = 4, ok = 200;
    if (this.readyState === done && this.status === ok) {
      my_JSON_object = this.response;
    }
  };
  xhttp.send();
  return my_JSON_object;
}
</script>
```

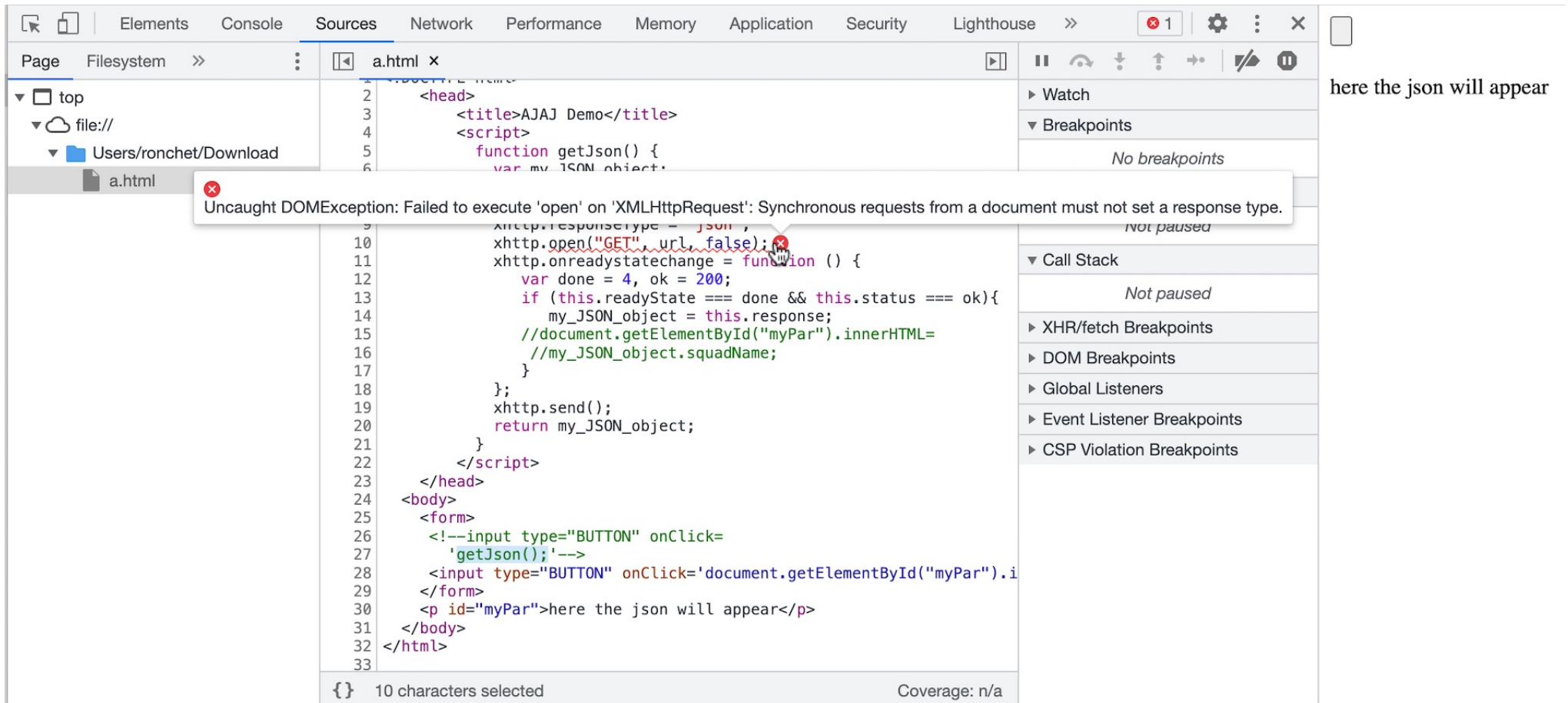
Let us make it sync

# The traps of asynchronous computing 2 out a

```
prova.html x
1 <!DOCTYPE html>
2 <head>
3   <title>Form Example</title>
4   <script>
5     function getJson() {
6
7     }
8     var xhttp = new XMLHttpRequest();
9     xhttp.open("GET", url, false);
10    //xhttp.responseType = "json";
11    xhttp.onreadystatechange = function () {
12      var done = 4, ok = 200;
13      if (this.readyState === done && this.status === ok)
14      {
```

Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end...

# The traps of asynchronous computing 2 out b



The screenshot shows a web browser's developer console with an error message: "Uncaught DOMException: Failed to execute 'open' on 'XMLHttpRequest': Synchronous requests from a document must not set a response type." The error is highlighted in a red box. The code snippet below the error shows an XMLHttpRequest object being used to make a synchronous request. The code is as follows:

```
2 <head>
3 <title>AJAJ Demo</title>
4 <script>
5   function getJson() {
6     var my_JSON_object;
7
8     xhttp.responseType = 'json';
9     xhttp.open("GET", url, false);
10    xhttp.onreadystatechange = function () {
11      var done = 4, ok = 200;
12      if (this.readyState === done && this.status === ok){
13        my_JSON_object = this.response;
14        //document.getElementById("myPar").innerHTML=
15        //my_JSON_object.squadName;
16      }
17    };
18    xhttp.send();
19    return my_JSON_object;
20  }
21 </script>
22 </head>
23 <body>
24 <form>
25 <!--input type="BUTTON" onClick=
26 'getJson();'-->
27 <input type="BUTTON" onClick='document.getElementById("myPar").i
28 </form>
29 <p id="myPar">here the json will appear</p>
30 </body>
31 </html>
32
33
```

On the right side of the console, there is a text box that says "here the json will appear".



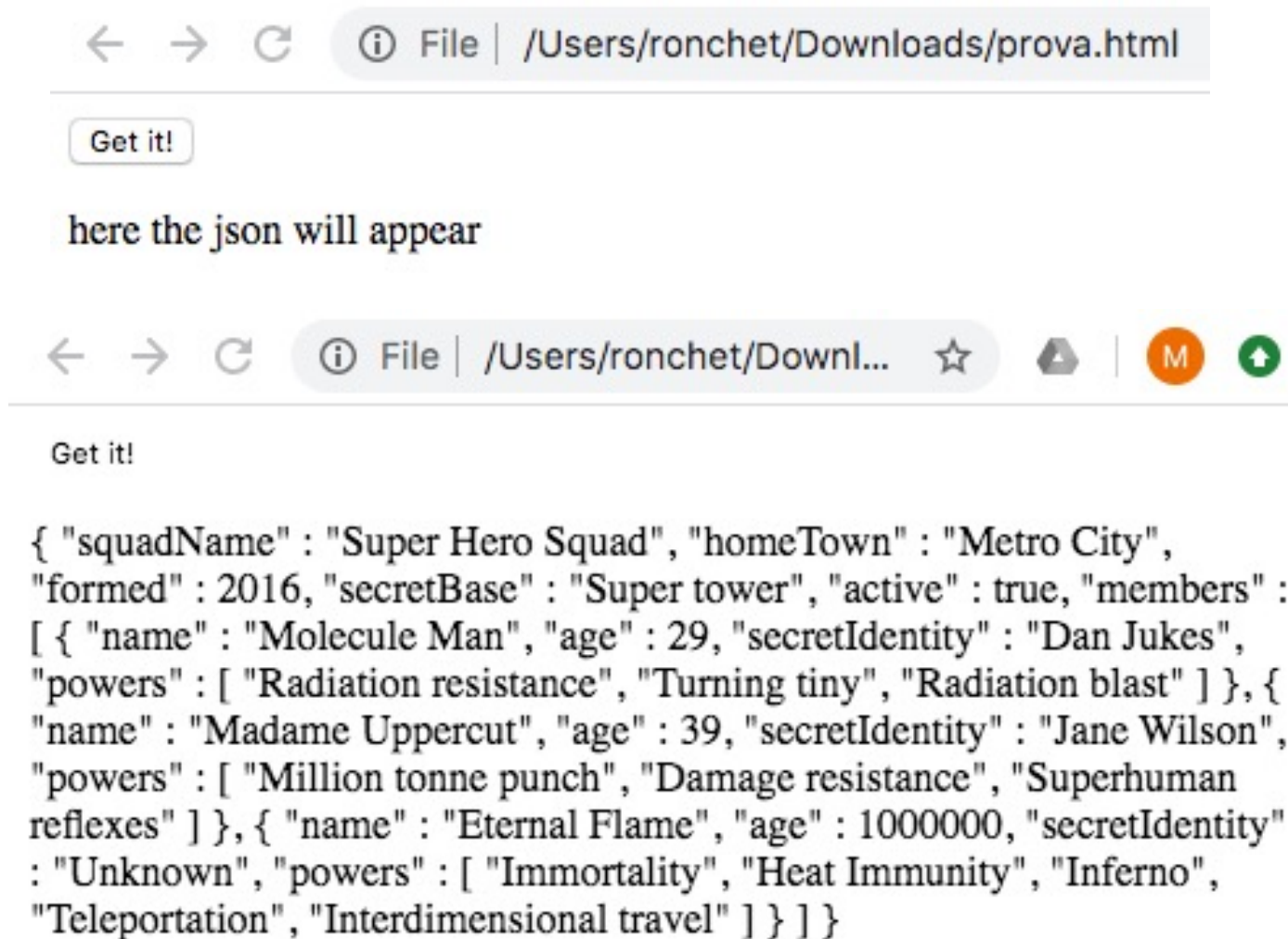
# The traps of asynchronous computing 3A

```
<script>
function getJson() {
  var my_JSON_object;
  var url="https://mdn.github.io/learning-
    area/javascript/oajs/json/superheroes.json"
  var xhttp = new XMLHttpRequest();
  xhttp.open("GET", url, false);
  //xhttp.responseType = "json";
  xhttp.onreadystatechange = function () {
    var done = 4, ok = 200;
    if (this.readyState === done && this.status === ok){
      my_JSON_object = this.response;
    }
  };
  xhttp.send();
  return my_JSON_object;
}
</script>
```

Let's comment this line



# The traps of asynchronous computing 3 out



The image shows two screenshots of a web browser. The top screenshot shows a browser window with the address bar containing the file path `/Users/ronchet/Downloads/prova.html`. Below the address bar is a button labeled "Get it!". The text "here the json will appear" is visible below the button. The bottom screenshot shows the same browser window after the button is clicked. The address bar now shows `/Users/ronchet/Downl...`. The "Get it!" button is still present. Below the button, a JSON object is displayed, representing the data returned by the asynchronous request.

```
{ "squadName" : "Super Hero Squad", "homeTown" : "Metro City",  
  "formed" : 2016, "secretBase" : "Super tower", "active" : true, "members" :  
  [ { "name" : "Molecule Man", "age" : 29, "secretIdentity" : "Dan Jukes",  
    "powers" : [ "Radiation resistance", "Turning tiny", "Radiation blast" ] }, {  
    "name" : "Madame Uppercut", "age" : 39, "secretIdentity" : "Jane Wilson",  
    "powers" : [ "Million tonne punch", "Damage resistance", "Superhuman  
reflexes" ] }, { "name" : "Eternal Flame", "age" : 1000000, "secretIdentity"  
: "Unknown", "powers" : [ "Immortality", "Heat Immunity", "Inferno",  
"Teleportation", "Interdimensional travel" ] } ] }
```

# The traps of asynchronous computing 4A

```
<script>
function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
        area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, true);
    xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
        var done = 4, ok = 200;
        if (this.readyState === done && this.status === ok) {
            my_JSON_object = this.response;
            document.getElementById("myPar").innerHTML=
                my_JSON_object.squadName;
        }
    };
    xhttp.send();
    return my_JSON_object;
}
</script>
```

Let's make it async again

This is the right way  
of doing things!

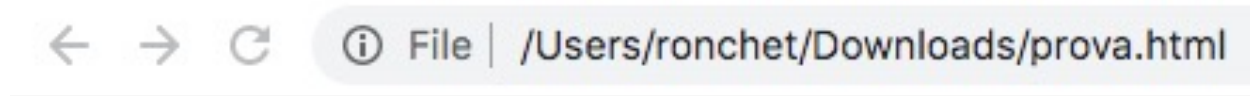


# The traps of asynchronous computing 4B

```
<!DOCTYPE html>
<head>
  <title>AJAJ Demo</title>
  <script>...</script>
</head>
<body>
  <form>
    <input type="BUTTON" onClick='getJson();'>
  </form>
  <p id="myPar">here the json will appear</p>
</body>
</html>
```

This is the right way  
of doing things!

# The traps of asynchronous computing 4 out



Get it!

here the json will appear



Get it!

Super Hero Squad

# The traps of asynchronous computing 5A

```
<script>
function getJson() {
  var my_JSON_object;
  var url="https://mdn.github.io/learning-
    area/javascript/oajs/json/superheroes.json"
  var xhttp = new XMLHttpRequest();
  xhttp.open("GET", url, true);
  //xhttp.responseType = "json";
  xhttp.onreadystatechange = function () {
    var done = 4, ok = 200;
    if (this.readyState === done && this.status === ok) {
      my_JSON_object = this.response;
      document.getElementById("myPar").innerHTML=
        my_JSON_object.squadName;
    }
  };
  xhttp.send();
  return my_JSON_object;
}
</script>
```

Let's comment this line

# The traps of asynchronous computing 5 out

← → ↻ ⓘ File | /Users/ronchet/Downloads/prova.html

Get it!

here the json will appear

← → ↻ ⓘ File | /Users/ronchet/Downloads/prova.html

Get it!

undefined

**Because my\_JSON\_Object is now a String!**

# The JavaScript JSON object

The **JSON** object contains methods for parsing JSON text, and converting values to JSON. It can't be called or constructed, has two method properties:

- **JSON.parse(*text*[, *reviver*])** Parse the string *text* as JSON, optionally transform the produced value and its properties, and return the value. Any violations of the JSON syntax, including those pertaining to the differences between JavaScript and JSON, cause a `SyntaxError` to be thrown. The *reviver* option allows for interpreting what the *replacer* has used to stand in for other datatypes.

# JSON.parse

JSON string => JS data

```
const json = '{"result":true, "count":42}';  
const obj = JSON.parse(json);  
  
console.log(obj.count);    // expected output: 42  
  
console.log(obj.result);  // expected output: true
```



# JSON.parse with reviver function

JSON string => JS data

```
var text = '{ "name": "Dorothea Wierer",  
             "birthDate": "1990-04-03",  
             "city": "Brunico" }';
```

```
var obj = JSON.parse(text, function (key, value) {  
    if (key == "birthDate") {  
        return new Date(value);  
    } else {  
        return value;  
    }  
});
```



reviver function

# JSON.stringify

JS data => JSON string

JS\_Object

```
console.log(JSON.stringify({ x: 5, y: 6 }));  
// expected output: '{"x":5,"y":6}'
```

JS\_Array

```
console.log(JSON.stringify([new Number(3), new String('false'),  
new Boolean(false)]));  
// expected output: "[3,\"false\",false]"
```

JS\_special types

```
console.log(JSON.stringify({ x: [10, undefined, function() {},  
Symbol('')] }));  
// expected output: '{"x":[10,null,null,null]}'
```

```
console.log(JSON.stringify(new Date(2006, 0, 2, 15, 4, 5)));  
// expected output: "\"2006-01-02T15:04:05.000Z\""
```

JS\_date

The JSON.stringify() function will remove any functions from a JavaScript object, both the key and the value

# Json Tutorial and reference

JS JSON

JSON Intro

JSON Syntax

JSON vs XML

JSON Data Types

JSON Parse

JSON Stringify

JSON Objects

JSON Arrays

JSON PHP

JSON HTML

JSON JSONP

[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>



# Q

**What is the most modern way to write AJAX (AJAJ) queries?**

# JS Promises

A **Promise** is a placeholder (proxy) for a value not known when the promise is created.

It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

In this way an asynchronous method returns a value like synchronous methods: it returns a *promise* to supply the value instead of returning the final value.

A very good tutorial: <https://javascript.info/promise-basics>

# JS Promises

- A Promise is in one of these states:
- *pending*: initial state (neither fulfilled nor rejected).
- *fulfilled*: the operation was completed successfully.
- *rejected*: the operation failed.

# JS Promises

Creating the Promise and evaluating it

```
let promise = new Promise(function(resolve, reject) {  
  ... some code...  
  if (ok) {  
    resolve(valueToBeReturned);  
  } else {  
    reject(new Error("message"));  
  }  
})
```

Any name, no implementation

# JS Request

```
promise.then(  
  function(value) {doSomethingWithValue(value);}, //if fulfilled  
  function(error) {doSomethingWithError(error);} //if rejected  
);
```



# JS Promises: example

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Promise</h2>
<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;
  // some code (try to change x to 5)
  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});
myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
</script>
</body>
</html>
```

shorter syntax

```
myPromise.then(
  myDisplayer,
  myDisplayer
);
```

try it here:

[https://www.w3schools.com/js/tryit.asp?filename=tryjs\\_promise2](https://www.w3schools.com/js/tryit.asp?filename=tryjs_promise2)



# JS Promises: example

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Promise</h2>
<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;
  // some code (try to change x to 5)
  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});
myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
</script>
</body>
</html>
```

```
    if (x == 0) {
      setTimeout(() => myResolve("OK"), 2000);
    } else {
      setTimeout(() => myReject("Error"), 1000);
    }
  });
```

with delay

try it here:

[https://www.w3schools.com/js/tryit.asp?filename=tryjs\\_promise2](https://www.w3schools.com/js/tryit.asp?filename=tryjs_promise2)



# JS fetch

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Fetch a file to change this text.</p>
<script>
let file = "fetch_info.txt"
fetch (file)
.then(x => x.text())
.then(y => document.getElementById("demo").innerHTML = y);
</script>
</body>
</html>
```



# JS fetch

fetch takes a Request object (may have headers)

fetch returns a Promise that resolve to a Response object

Response has methods:

- text() returns a Promise that resolves to a string
- json() returns a Promise that resolves to a Json object
- blob() returns a Promise that resolves to a Blob object