Lecture 9
Talking to The Web: Extroverted Apps

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hknu.ac.kr/

Mighty Gumball wants a Web app

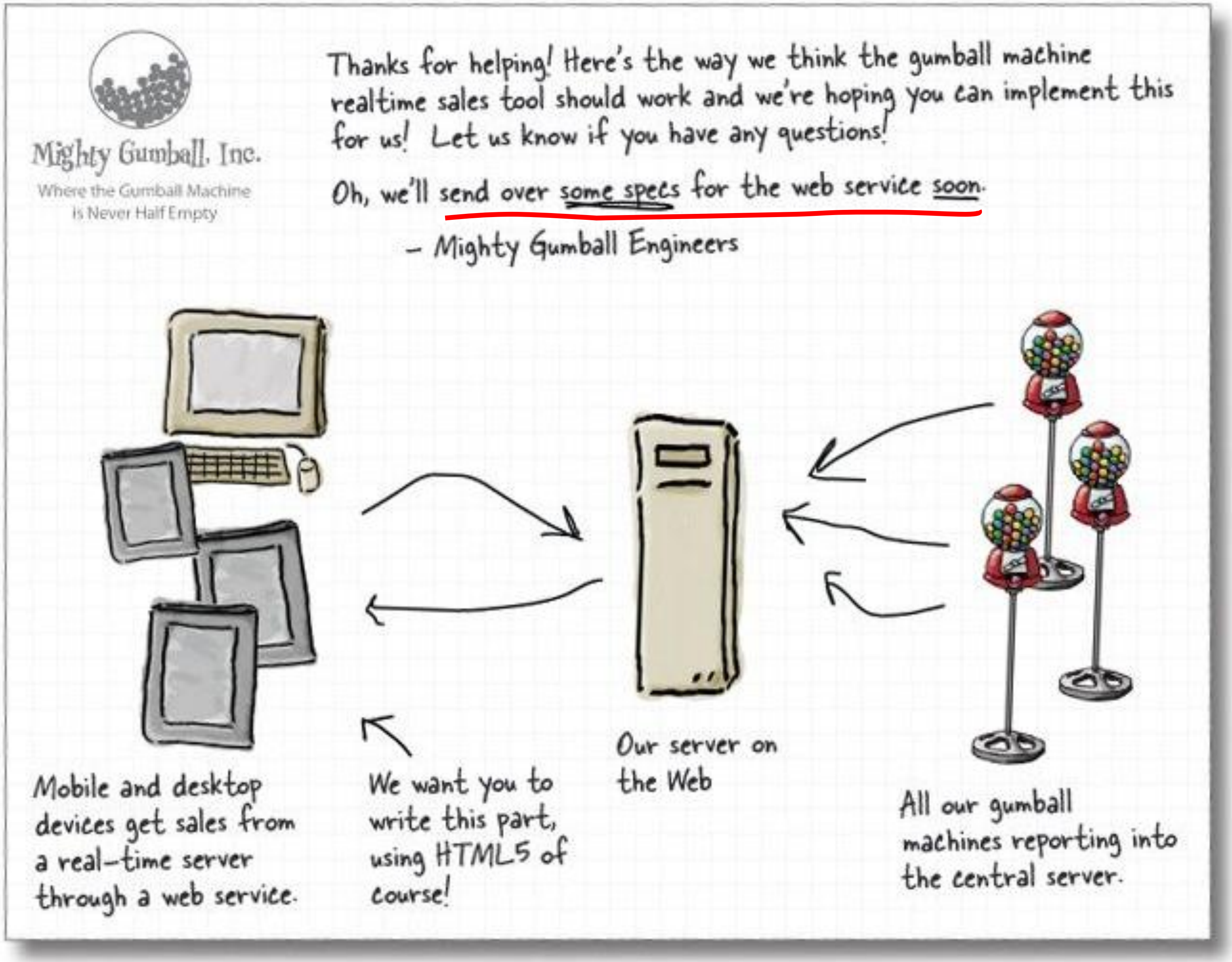
This just in: **Mighty Gumball**, Inc., an innovative company that builds and deploys real gumball machines, has contacted us for some help.
If you're not up on them, they've recently **network-enabled their gumball machines to track sales in near real time.**

Now it almost goes without saying that Mighty Gumball are gumball experts, not software developers, and

so they'd like our help **building an app to help them monitor gumball sales.**

Check out the new Web-enabled MG2200 gumball machine. It's going to revolutionize the biz.
CEO, MightyGumball

Mighty Gumball wants a Web app



Thanks for helping! Here's the way we think the gumball machine realtime sales tool should work and we're hoping you can implement this for us! Let us know if you have any questions!

Oh, we'll send over some specs for the web service soon.

– Mighty Gumball Engineers

Mobile and desktop devices get sales from a real-time server through a web service.

We want you to write this part, using HTML5 of course!

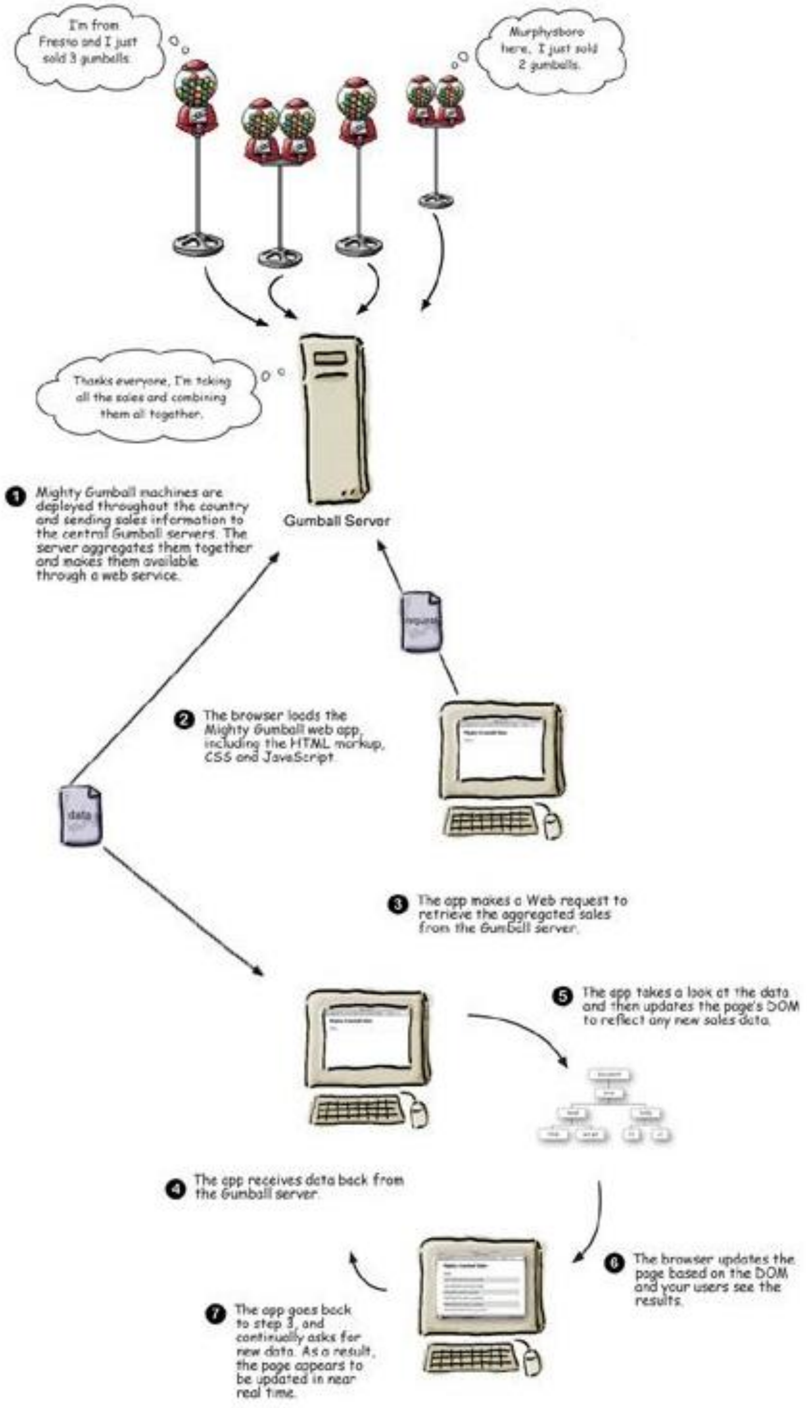
Our server on the Web

All our gumball machines reporting into the central server.

A little more background on Mighty Gumball

You probably need a little background beyond Mighty Gumball’s short note. Here’s what we’ve got:

- ✓ First, they’ve got gumball machines all over the country sending sales reports to a Mighty Gumball server, which combines **all those reports** and makes them available **through a web service**
- ✓ Second, they’re asking us to **build a web app that displays the sales in a browser for the Gumball Sales team.** And, most likely they want this report to be updated as the sales change over time


I'm from Fresno and I just sold 3 gumballs.
Murphysboro here, I just sold 2 gumballs.
Thanks everyone, I'm taking all the sales and combining them all together.
Gumball Server
request
data

1. Mighty Gumball machines are deployed throughout the country and sending sales information to the central Gumball servers. The server aggregates them together and makes them available through a web service.

2. The browser loads the Mighty Gumball web app, including the HTML markup, CSS and JavaScript.

3. The app makes a Web request to retrieve the aggregated sales from the Gumball server.

4. The app receives data back from the Gumball server.

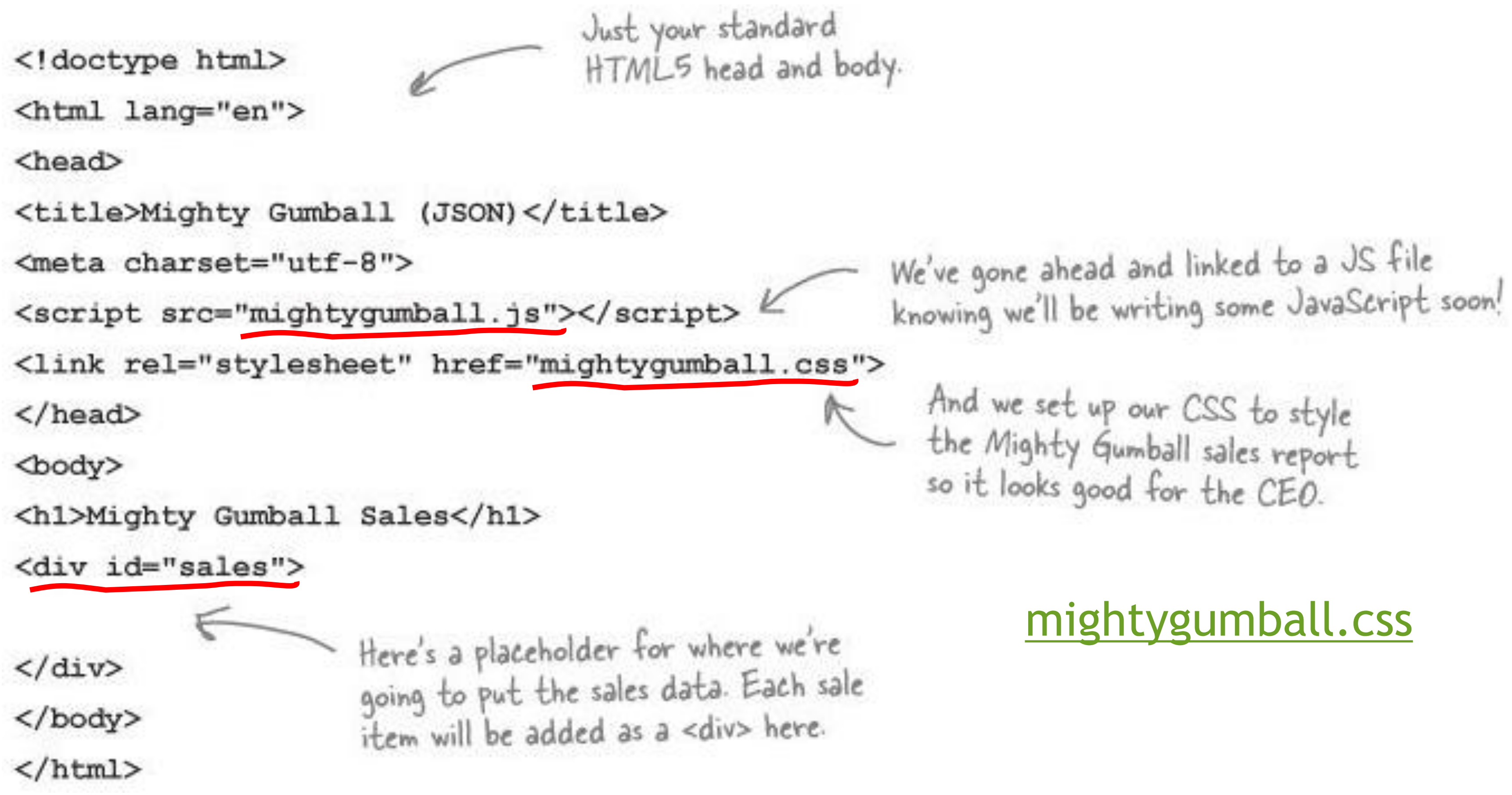
5. The app takes a look at the data and then updates the page's DOM to reflect any new sales data.

6. The browser updates the page based on the DOM and your users see the results.

7. The app goes back to step 3, and continually asks for new data. As a result, the page appears to be updated in near real time.

Just a quick start…

- All we need is a **place to put our sales reports** as they come in
- Will take a look at **how to retrieve things via the Web**



```
<!doctype html>
<html lang="en">
<head>
<title>Mighty Gumball (JSON)</title>
<meta charset="utf-8">
<script src="mightygumball.js"></script>
<link rel="stylesheet" href="mightygumball.css">
</head>
<body>
<h1>Mighty Gumball Sales</h1>
<div id="sales">

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

Just your standard HTML5 head and body.

We've gone ahead and linked to a JS file knowing we'll be writing some JavaScript soon!

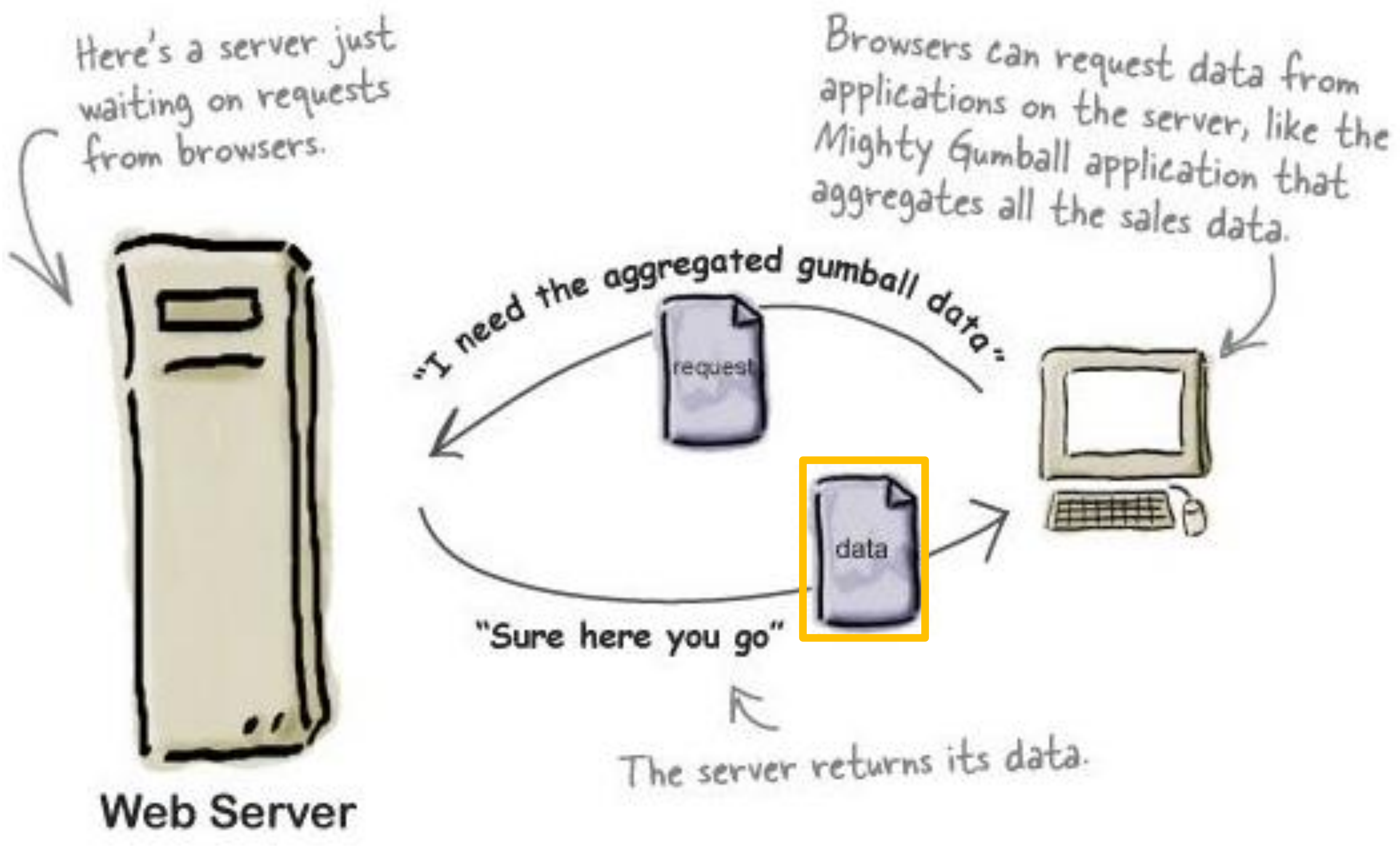
And we set up our CSS to style the Mighty Gumball sales report so it looks good for the CEO.

Here's a placeholder for where we're going to put the sales data. Each sale item will be added as a <div> here.

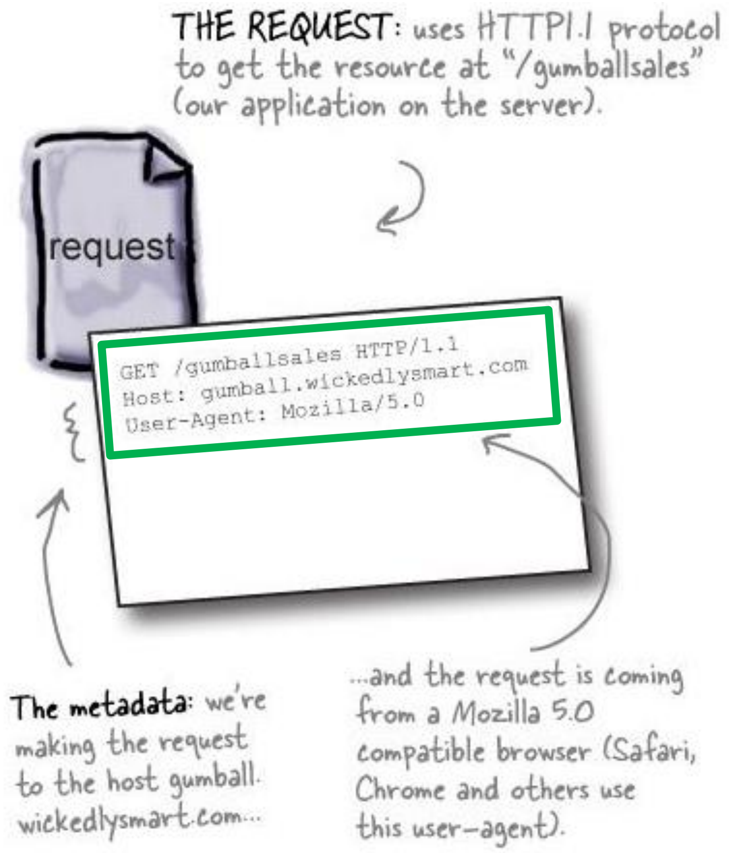
mightygumball.css

So how do we make requests to web services?

- Knows **how a browser requests a page from a web server**
- Browser can also **retrieve data with HTTP from a web server** in the same way


Here's a server just waiting on requests from browsers.
Browsers can request data from applications on the server, like the Mighty Gumball application that aggregates all the sales data.
"I need the aggregated gumball data"
request
data
"Sure here you go"
The server returns its data.
Web Server

The request takes care of telling the server what data we're after, while the response contains **metadata** and the **data we requested:**

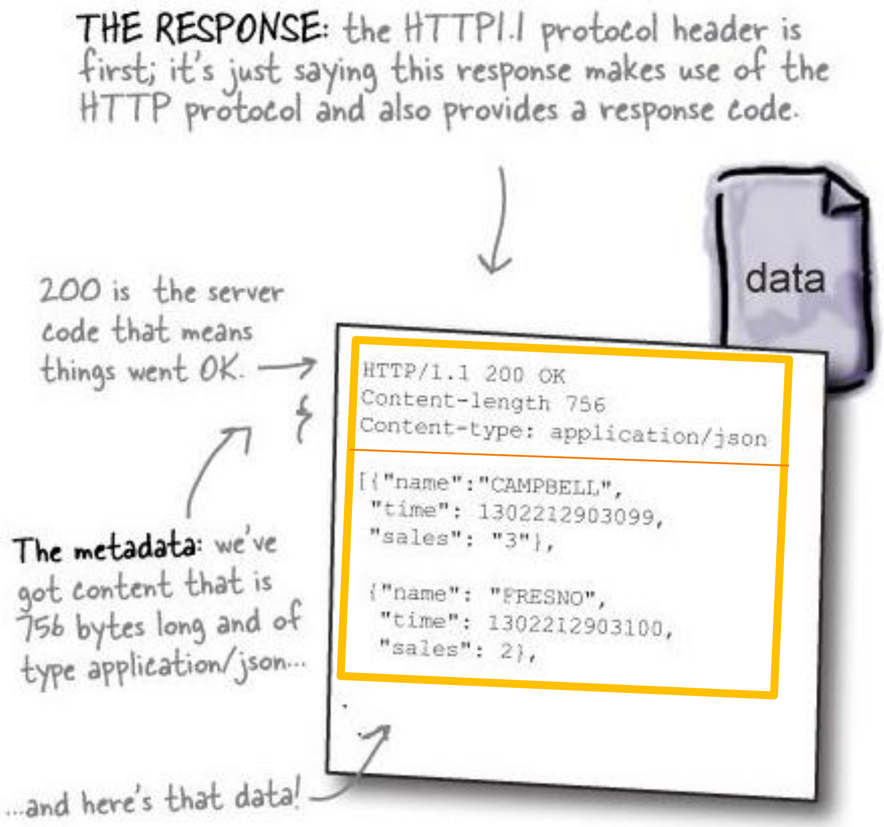


THE REQUEST: uses HTTP1.1 protocol to get the resource at "/gumballsales" (our application on the server).

```
GET /gumballsales HTTP/1.1
Host: gumball.wickedlysmart.com
User-Agent: Mozilla/5.0
```

The metadata: we're making the request to the host gumball.wickedlysmart.com...

...and the request is coming from a Mozilla 5.0 compatible browser (Safari, Chrome and others use this user-agent).



THE RESPONSE: the HTTP1.1 protocol header is first; it's just saying this response makes use of the HTTP protocol and also provides a response code.

200 is the server code that means things went OK.

```
HTTP/1.1 200 OK
Content-length 756
Content-type: application/json

[{"name":"CAMPBELL",
 "time": 1302212903099,
 "sales": "3"},

 {"name": "FRESNO",
  "time": 1302212903100,
  "sales": 2},
```

The metadata: we've got content that is 756 bytes long and of type application/json...

...and here's that data!

Note: This pattern of retrieving data using **XMLHttpRequest** is commonly referred to as **"Ajax"** or **XHR.**

How to make a request from JavaScript

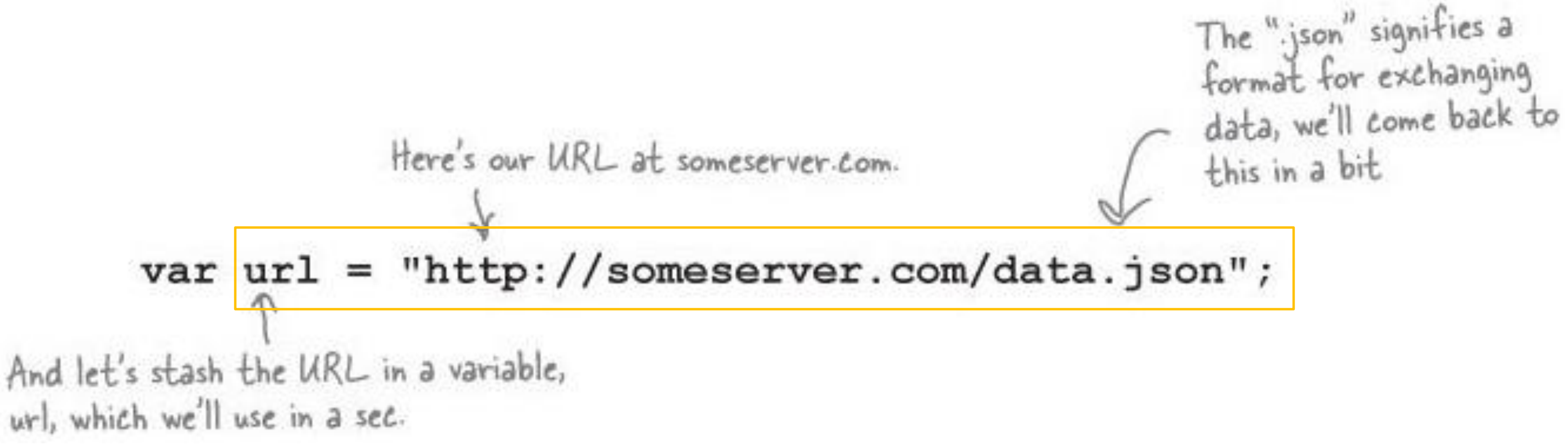
How can retrieve data with HTTP?

- Writes a little code to create an actual HTTP request
- Asks the browser to make the request *on our behalf*
- After it's made the request, the browser will then hand us back the data it receives.

Let's step through making an HTTP request:

How can retrieve data with HTTP?

1. <u>Starts with a URL</u>
   After all, we need to **tell the browser where to get the data** we're after:



Here's our URL at someserver.com.

The ".json" signifies a format for exchanging data, we'll come back to this in a bit

```
var url = "http://someserver.com/data.json";
```

And let's stash the URL in a variable, url, which we'll use in a sec.

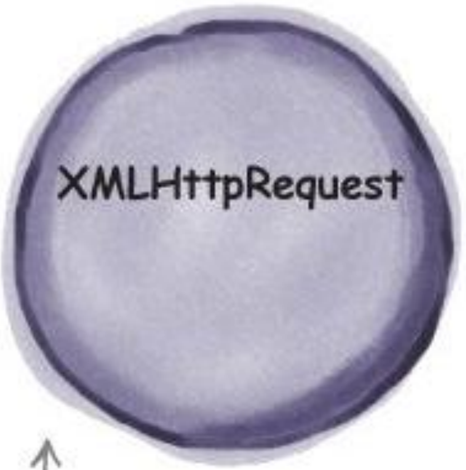
How can retrieve data with HTTP?

2. Next we'll create a request object, like this:

```
var request = new XMLHttpRequest();
```

We're assigning the request object to the variable request.

And we use the XMLHttpRequest constructor to create a new request object. We'll talk about the "XML" part of that in a bit

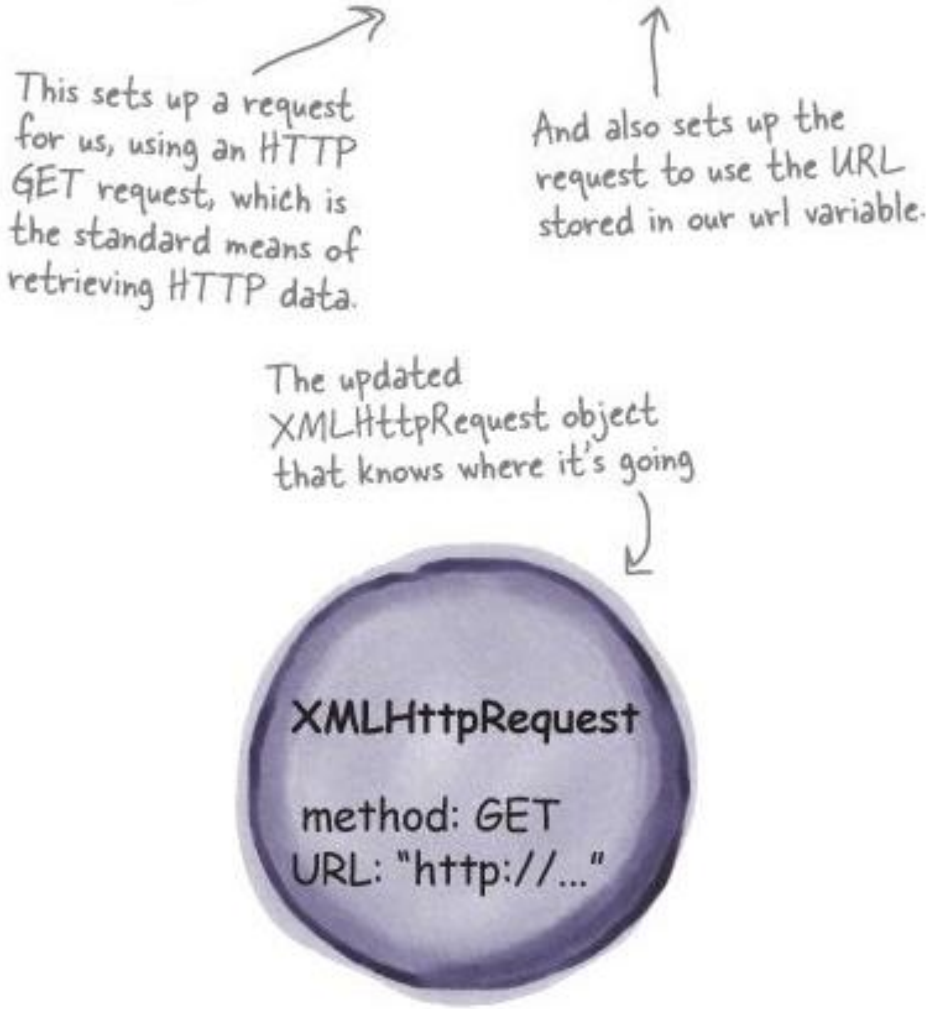


A brand new XMLHttpRequest object.

How can retrieve data with HTTP?

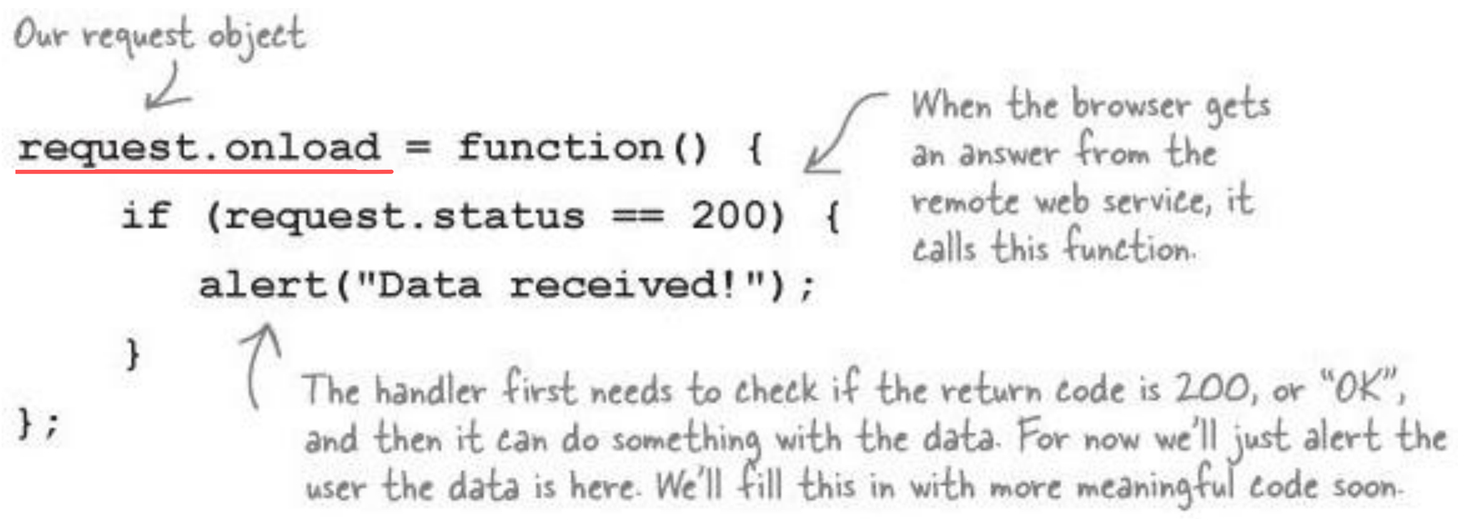
3. Uses the request object's open method.

```
request.open("GET", url);
```


This sets up a request for us, using an HTTP GET request, which is the standard means of retrieving HTTP data.
And also sets up the request to use the URL stored in our url variable.
The updated XMLHttpRequest object that knows where it's going
XMLHttpRequest
method: GET
URL: "http://..."

How can retrieve data with HTTP?

4. Provides a handler that is called when the data arrives.

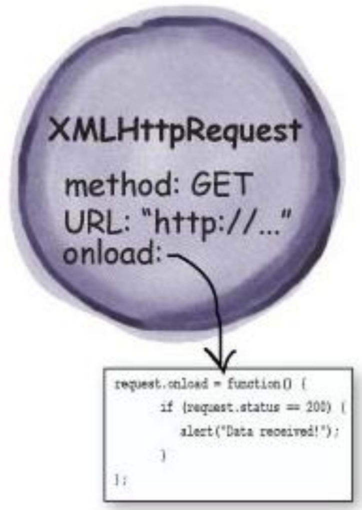


Our request object

```
request.onload = function() {
    if (request.status == 200) {
        alert("Data received!");
    }
};
```

When the browser gets an answer from the remote web service, it calls this function.

The handler first needs to check if the return code is 200, or "OK", and then it can do something with the data. For now we'll just alert the user the data is here. We'll fill this in with more meaningful code soon.



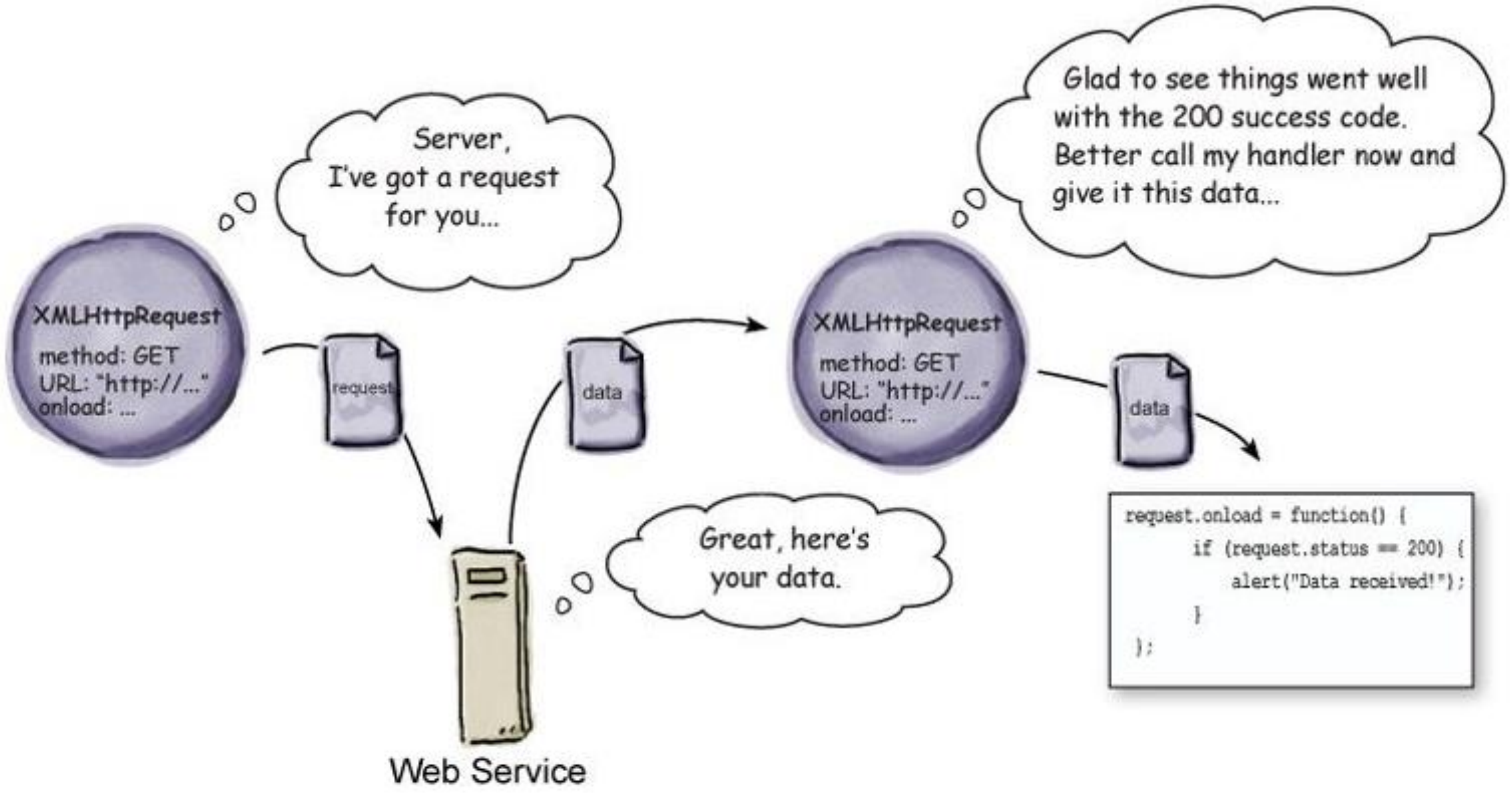
```
request.onload = function() {
    if (request.status == 200) {
        alert("Data received!");
    }
};
```

How can retrieve data with HTTP?

5. Tells the request to go out and get the data, and to do that we use the send method:

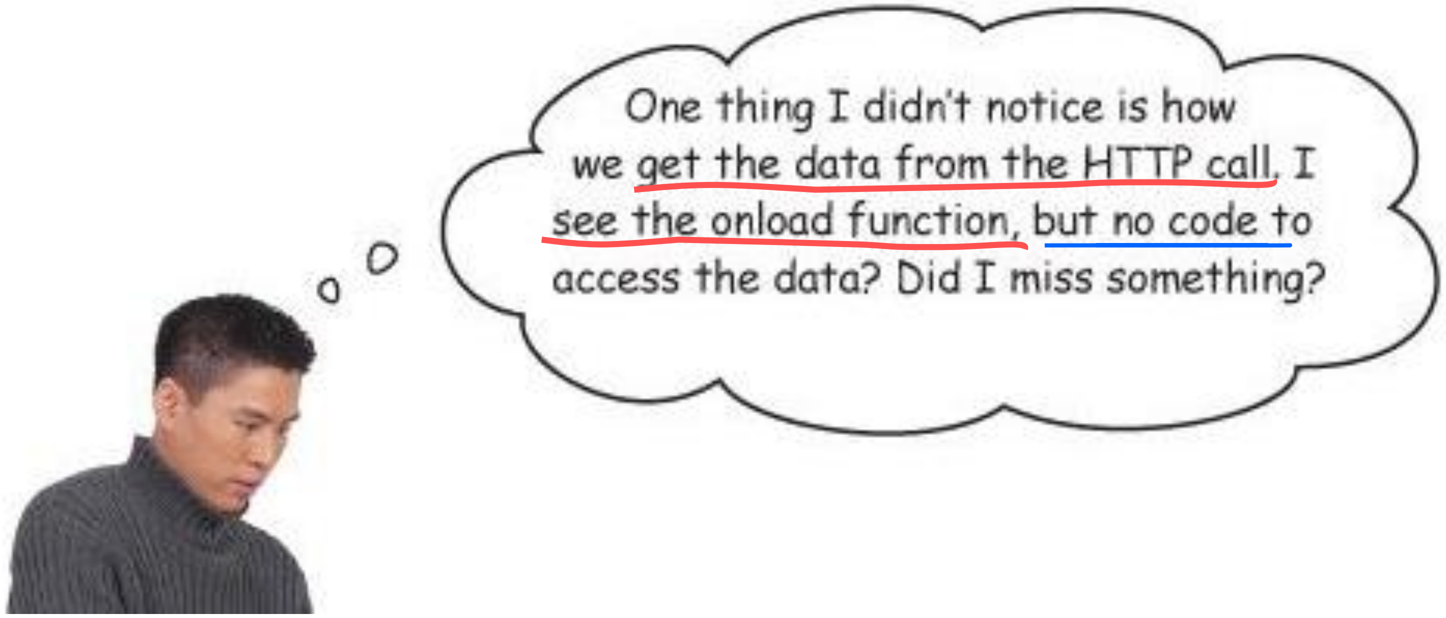
```
request.send(null);
```

This sends the request to the server. We pass null if we're not sending any data to the remote service (which we're not).

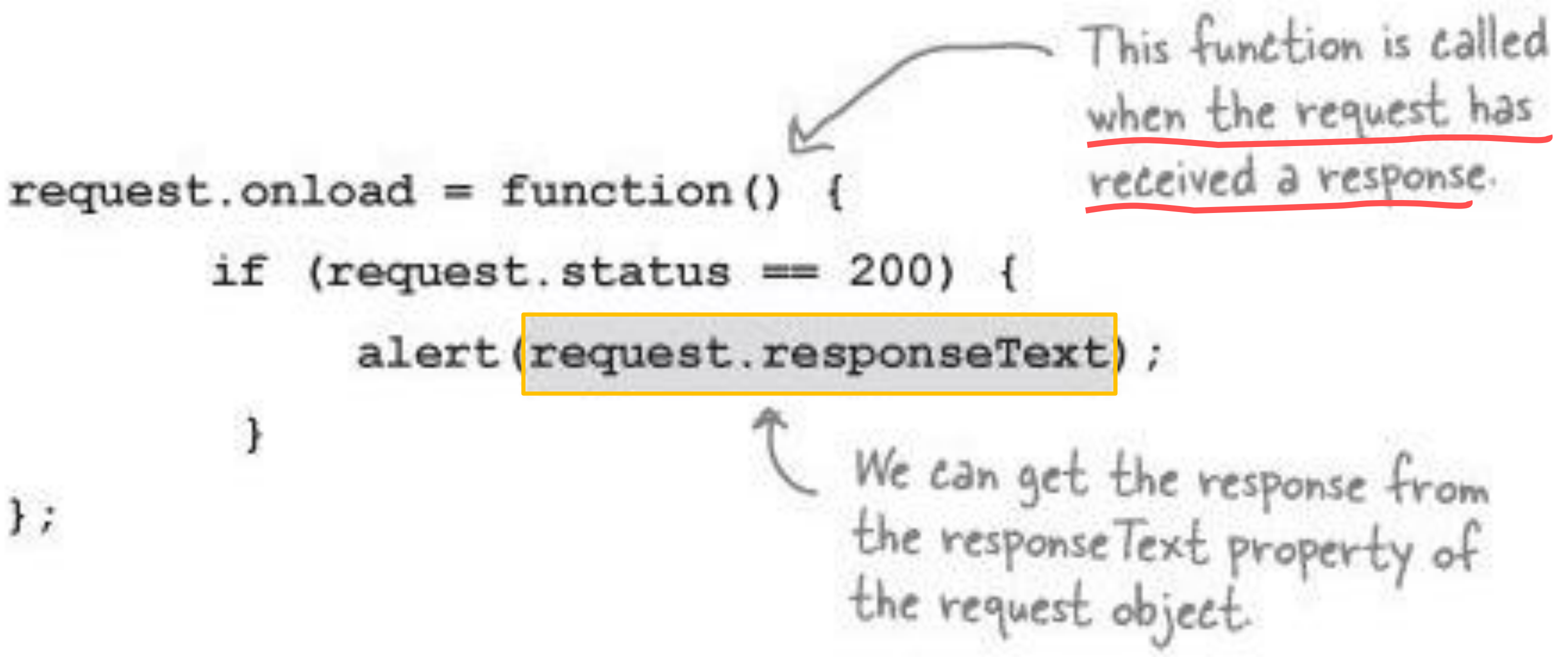


```
request.onload = function() {
        if (request.status == 200) {
            alert("Data received!");
        }
};
```

So, to review: we create an XMLHttpRequest object, load it with a URL and HTTP request type, along with a handler. Then we send the request and wait for the data to arrive. When it does, the handler is called.


One thing I didn't notice is how we get the data from the HTTP call. I see the onload function, but no code to access the data? Did I miss something?

We just hadn’t quite got there yet. The data from the **HTTP GET retrieval** can be found in the responseText property of the request object. So we can write code like this:



```
request.onload = function() {
        if (request.status == 200) {
              alert(request.responseText);
        }
};
```

This function is called when the request has received a response.

We can get the response from the responseText property of the request object.

See 제 08강 XML Tutorial [2].ppt pp. 41-45

실습과제 09-1 Code Magnets

```
window.onload = function () {
    var url = "http://wickedlysmart.com/ifeelluckytoday";
    var request = ____________________
    ______________________________ {
        if (____________________) {
            displayLuck(____________________);
        }
    };
    ______________________________
    ________________________
}
function displayLuck(luck) {
    var p = document.____________________("luck");
    p.__innerHTML__ = "Today you are " + luck;
}
```

Your magnets go here!

코딩하지 말고 그냥 적어내면 됨!

```
new TextHttpRequest();
request.create("GET", url);
var i = 0;
request.responseText
request.send(null);
request.open("GET", url);
request.onload = function()
myLuckyText
new XMLHttpRequest();
getElementById
request.status == 200
```

Move over XML, meet JSON

XML was going to save us all — a **data format that was human readable and machine parseable**, a data format that was going to support all the data needs of the world

When XMLHttpRequest was first developed, XML was indeed the way we all exchanged data *(thus, the name XMLHttpRequest).*

"Well, along the way XML apparently slipped on a banana peel thrown by JSON."

Who's JSON?

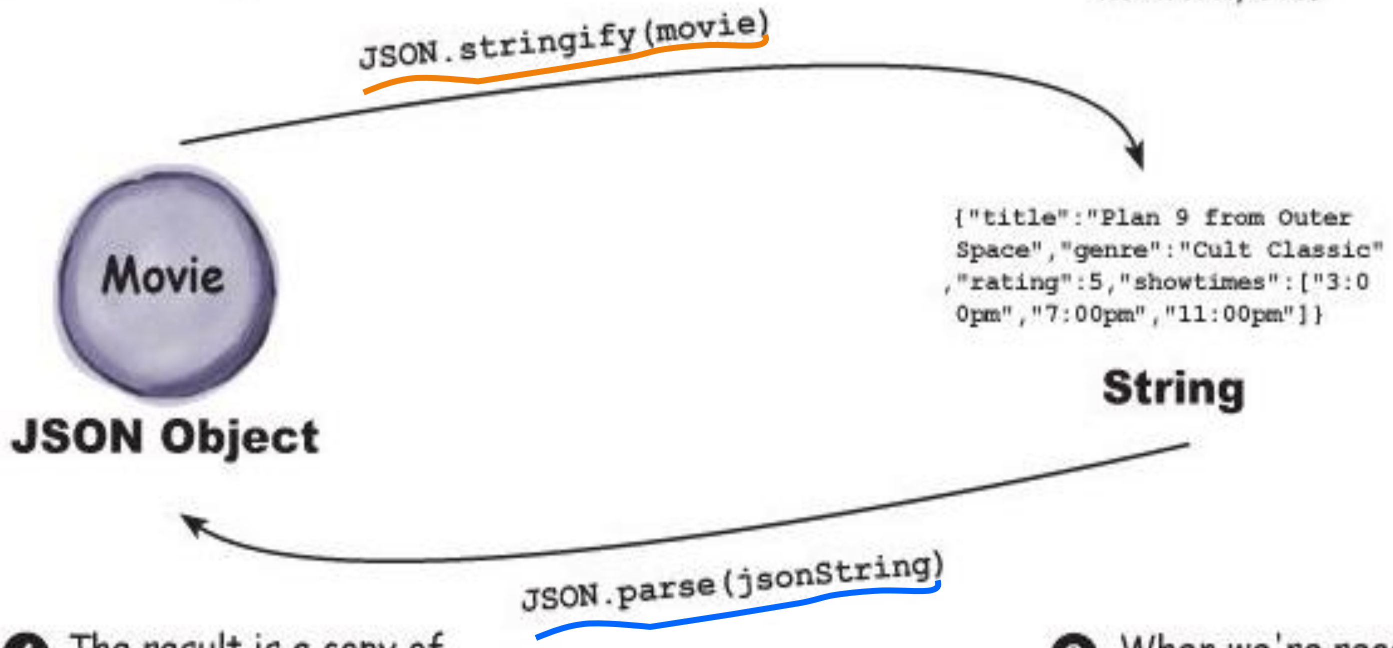
Just the **latest and greatest data format**, born out of JavaScript, and being adopted across the Web in the browser and on the server side.

What's so great about JSON?

- Well, it's pretty darn human-readable, and it can be **parsed quickly and easily straight into JavaScript values and objects**
- Unlike XML, it's so cute
- **Exchanges JavaScript data over the network**, to store data in a local store with the Web Storage API, and as part of another way to access web data

❶ We have a JavaScript object we want to exchange or store, so we call the JSON.stringify method, passing the object as the argument.

❷ The result is a string that represents the object. We can store this string, pass it to a function, send it over the network, etc.



```
{"title":"Plan 9 from Outer
Space","genre":"Cult Classic"
,"rating":5,"showtimes":["3:0
0pm","7:00pm","11:00pm"]}
```

❸ When we're ready to turn the string back into an object, we pass it to the JSON.parse method.

❹ The result is a copy of our original object.

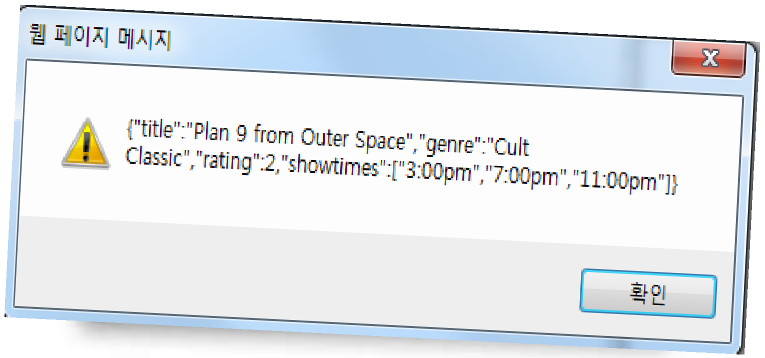
A quick example using JSON

Let's run through a quick example that **converts an object into its JSON string format.**

```
var plan9Movie = new Movie("Plan 9 from Outer Space","Cult Classic", 2,
                               ["3:00pm", "7:00pm", "11:00pm"]);
```

Here's a nice movie object complete with strings, numbers and an array.

1. Once you've got an object, you can convert it into the JSON string format with the JSON.stringify method.



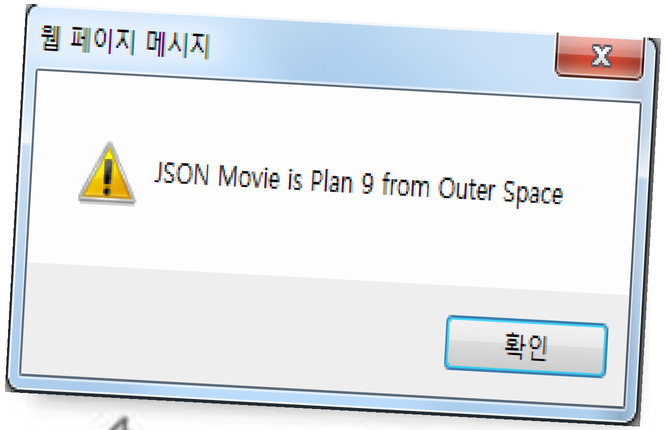
웹 페이지 메시지

```
{"title":"Plan 9 from Outer Space","genre":"Cult Classic","rating":2,"showtimes":["3:00pm","7:00pm","11:00pm"]}
```

```
var jsonString = JSON.stringify(plan9Movie);
alert(jsonString);
```

Here's the result, a string version of the object displayed in the alert.

2. How would we turn it back into an object we can do something with?
   Just use **JSON.stringify**'s sister method: JSON.parse

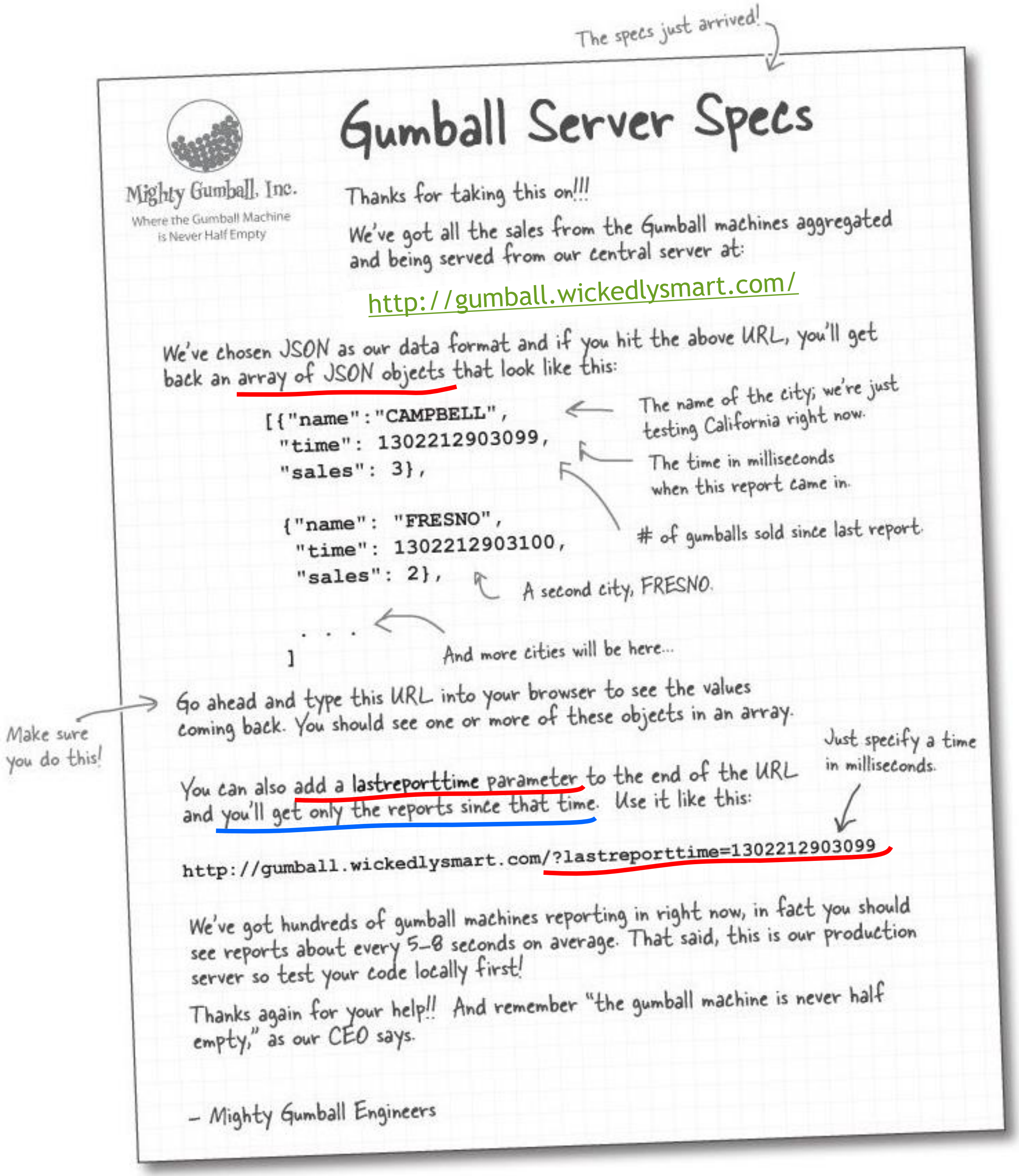


JSON Movie is Plan 9 from Outer Space

```
var jsonMovieObject = JSON.parse(jsonString);
alert("JSON Movie is " + jsonMovieObject.title);
```

Ah, and now we use this as a real object, accessing its properties.

http://www.json.org/



The specs just arrived!

Mighty Gumball, Inc.
Where the Gumball Machine is Never Half Empty

Gumball Server Specs

Thanks for taking this on!!!

We've got all the sales from the Gumball machines aggregated and being served from our central server at:

http://gumball.wickedlysmart.com/

We've chosen JSON as our data format and if you hit the above URL, you'll get back an array of JSON objects that look like this:

```
[{"name":"CAMPBELL",
  "time": 1302212903099,
  "sales": 3},

  {"name":  "FRESNO",
   "time": 1302212903100,
   "sales": 2},

     . . .
 ]
```

The name of the city; we're just testing California right now.

The time in milliseconds when this report came in.

# of gumballs sold since last report.

A second city, FRESNO.

And more cities will be here...

Make sure you do this!

Go ahead and type this URL into your browser to see the values coming back. You should see one or more of these objects in an array.

You can also add a lastreporttime parameter to the end of the URL and you'll get only the reports since that time. Use it like this:

Just specify a time in milliseconds.

```
http://gumball.wickedlysmart.com/?lastreporttime=1302212903099
```

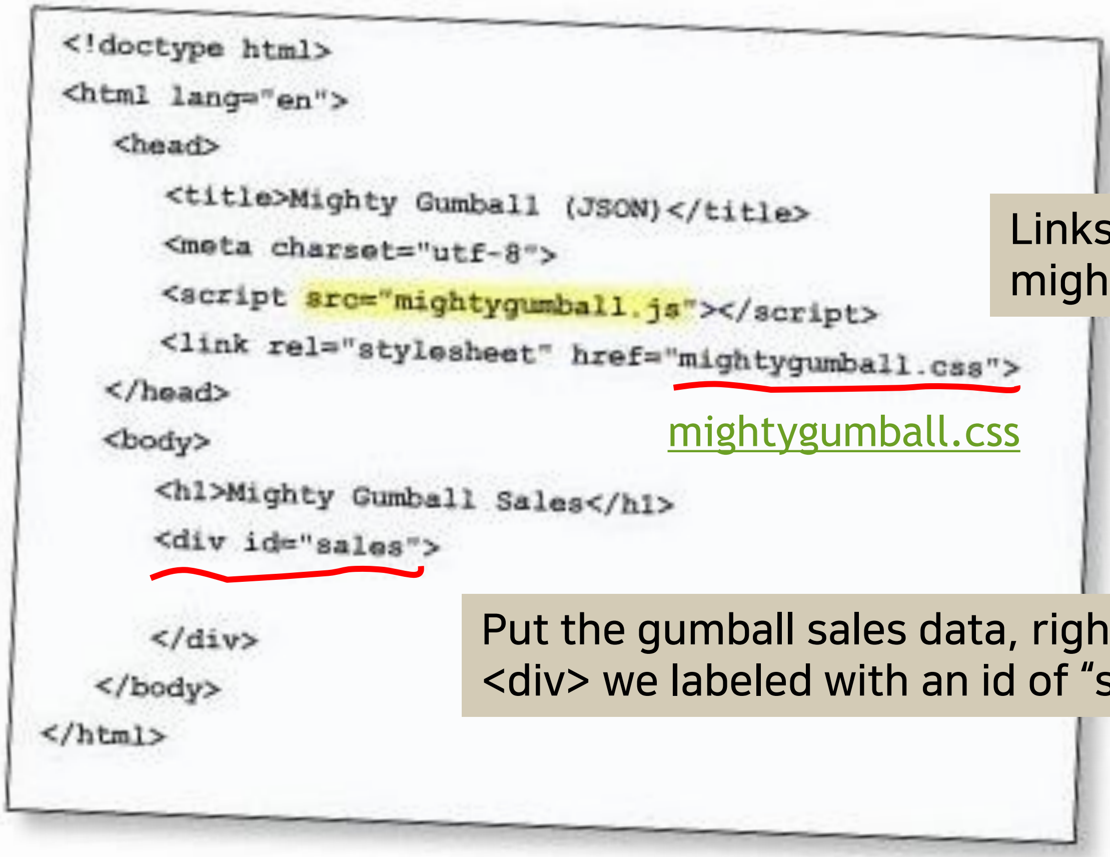
We've got hundreds of gumball machines reporting in right now, in fact you should see reports about every 5-8 seconds on average. That said, this is our production server so test your code locally first!

Thanks again for your help!! And remember "the gumball machine is never half empty," as our CEO says.

– Mighty Gumball Engineers

Let's get to work!

- Have done your training on XMLHttpRequest and JSON
- All ready to get some code written and to get a first cut of the Gumball App running



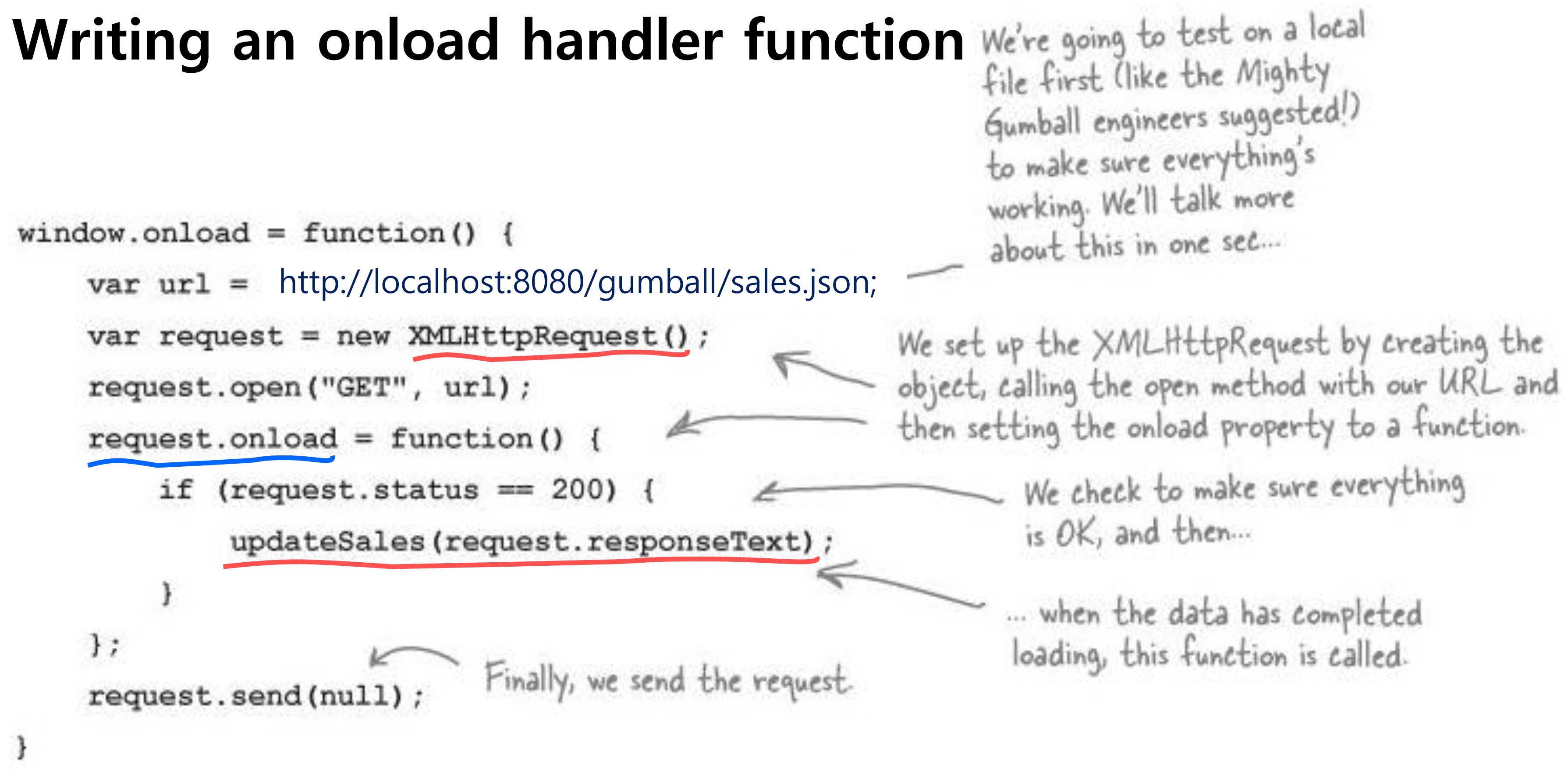
```
<!doctype html>
<html lang="en">
    <head>
        <title>Mighty Gumball (JSON)</title>
        <meta charset="utf-8">
        <script src="mightygumball.js"></script>
        <link rel="stylesheet" href="mightygumball.css">
    </head>
    <body>
        <h1>Mighty Gumball Sales</h1>
        <div id="sales">

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

Links to a file called mightygumball.js

mightygumball.css

Put the gumball sales data, right into the <div> we labeled with an id of "sales"

Writing an onload handler function



```
window.onload = function() {
    var url =  http://localhost:8080/gumball/sales.json;
    var request = new XMLHttpRequest();
    request.open("GET", url);
    request.onload = function() {
        if (request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.send(null);
}
```

We're going to test on a local file first (like the Mighty Gumball engineers suggested!) to make sure everything's working. We'll talk more about this in one sec...

We set up the XMLHttpRequest by creating the object, calling the open method with our URL and then setting the onload property to a function.

We check to make sure everything is OK, and then...

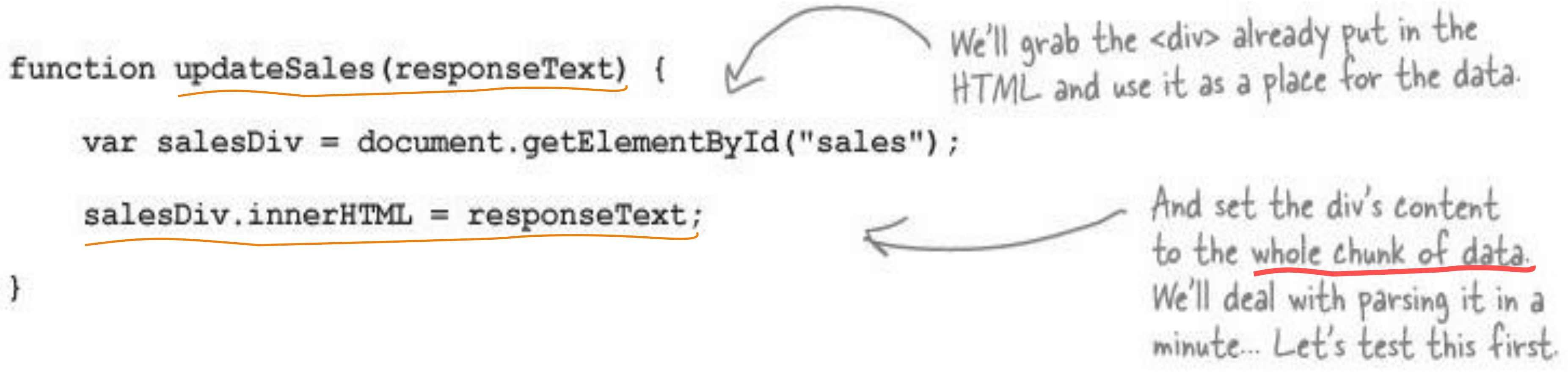
... when the data has completed loading, this function is called.

Finally, we send the request.

- Writes an **onload handler** that gets invoked when the HTML is fully loaded
- Fires off an HTTP request to get the sales data
- Asks the XMLHttpRequest to call the function updateSales

Displaying the gumball sales data

- Writes the handler, updateSales
- Just goes with the simplest implementation possible, we can always make it better later:



```
function updateSales(responseText) {
    var salesDiv = document.getElementById("sales");
    salesDiv.innerHTML = responseText;
}
```

We'll grab the <div> already put in the HTML and use it as a place for the data.

And set the div's content to the whole chunk of data. We'll deal with parsing it in a minute... Let's test this first.

Watch Out, Detour Ahead!

- Has a little detour to take care of first
- Tests locally before hitting their production server
- But, need the data to live on a server so that XMLHttpRequest can use the HTTP protocol to retrieve it

In terms of servers you've got a few choices:

1. If your company has servers that are available for testing, use those.

2. Or, you can use a **third-party hosting service** like Dothome, Dreamhost or one of many other hosting companies.

3. Finally, you can set up a server right on your **own machine**.
   In that case your URLs are going to look something like:

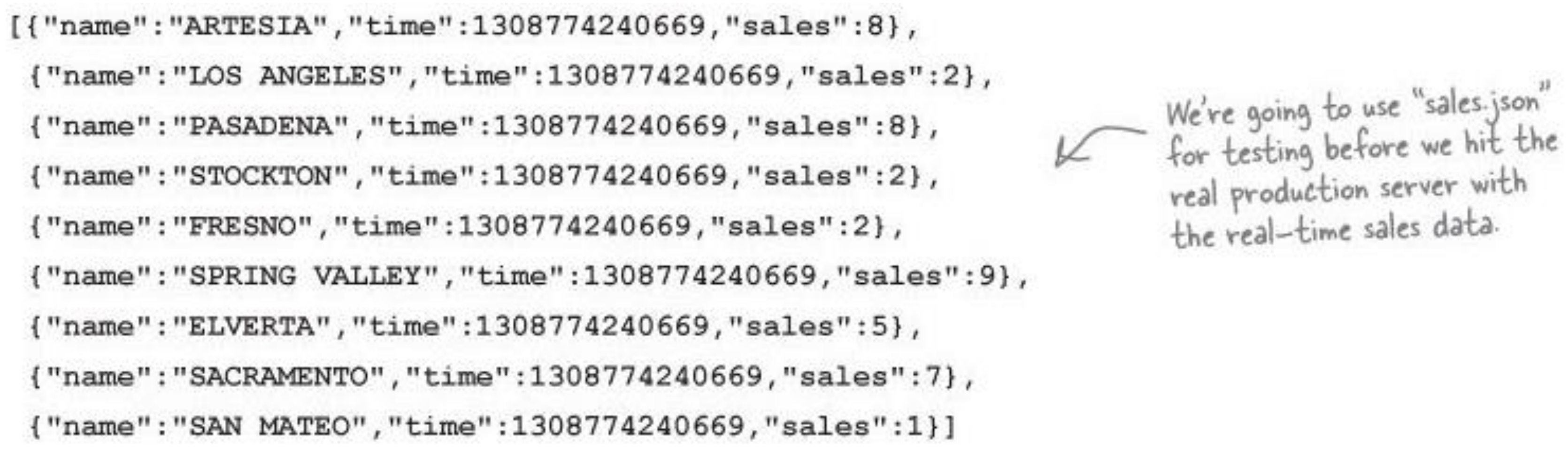
   http://localhost:8080/gumball/mightygumball.html

How to set up your own Web Server

Tomcat Server

- ✓ Copy ‘**sales.json**’ to ‘**C:\apache-tomcat-7.0.55\webapps\gumball\**’

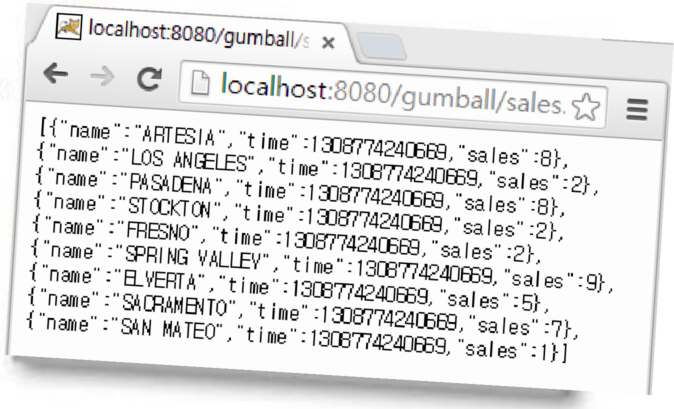
sales.json sales.json



```
[{"name":"ARTESIA","time":1308774240669,"sales":8},
 {"name":"LOS ANGELES","time":1308774240669,"sales":2},
 {"name":"PASADENA","time":1308774240669,"sales":8},
 {"name":"STOCKTON","time":1308774240669,"sales":2},
 {"name":"FRESNO","time":1308774240669,"sales":2},
 {"name":"SPRING VALLEY","time":1308774240669,"sales":9},
 {"name":"ELVERTA","time":1308774240669,"sales":5},
 {"name":"SACRAMENTO","time":1308774240669,"sales":7},
 {"name":"SAN MATEO","time":1308774240669,"sales":1}]
```

We're going to use "sales.json" for testing before we hit the real production server with the real-time sales data.

Back to the code



```
[{"name":"ARTESIA","time":1308774240669,"sales":8},
{"name":"LOS ANGELES","time":1308774240669,"sales":2},
{"name":"PASADENA","time":1308774240669,"sales":8},
{"name":"STOCKTON","time":1308774240669,"sales":2},
{"name":"FRESNO","time":1308774240669,"sales":2},
{"name":"SPRING VALLEY","time":1308774240669,"sales":9},
{"name":"ELVERTA","time":1308774240669,"sales":5},
{"name":"SACRAMENTO","time":1308774240669,"sales":7},
{"name":"SAN MATEO","time":1308774240669,"sales":1}]
```

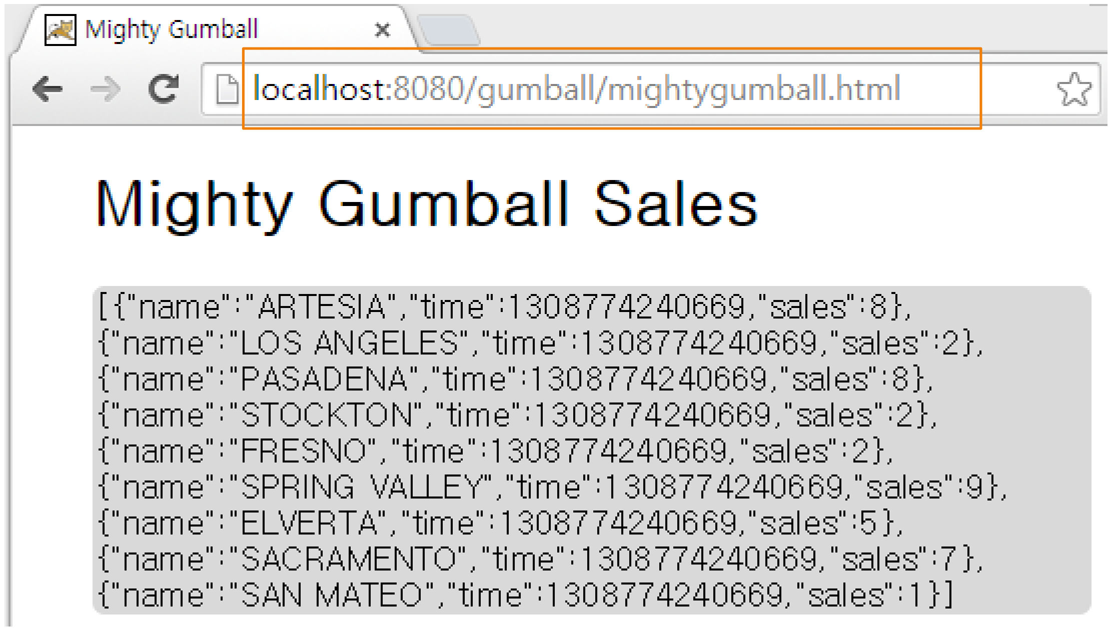
It helps to first test this URL in your browser to make sure it works.

```
window.onload = function() {
    var url = http://localhost:8080/gumball/sales.json;
    var request = new XMLHttpRequest();
    request.open("GET", url);
    request.onload = function() {
        if (request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.send(null);
}
```

Make sure this is pointing to the right URL.

Let's test this already!

Remember we're sending an HTTP request to get the data in sales.json, which we're just dumping into the <div> for now. Looks like it worked!



Mighty Gumball Sales

```
[{"name":"ARTESIA","time":1308774240669,"sales":8},
{"name":"LOS ANGELES","time":1308774240669,"sales":2},
{"name":"PASADENA","time":1308774240669,"sales":8},
{"name":"STOCKTON","time":1308774240669,"sales":2},
{"name":"FRESNO","time":1308774240669,"sales":2},
{"name":"SPRING VALLEY","time":1308774240669,"sales":9},
{"name":"ELVERTA","time":1308774240669,"sales":5},
{"name":"SACRAMENTO","time":1308774240669,"sales":7},
{"name":"SAN MATEO","time":1308774240669,"sales":1}]
```

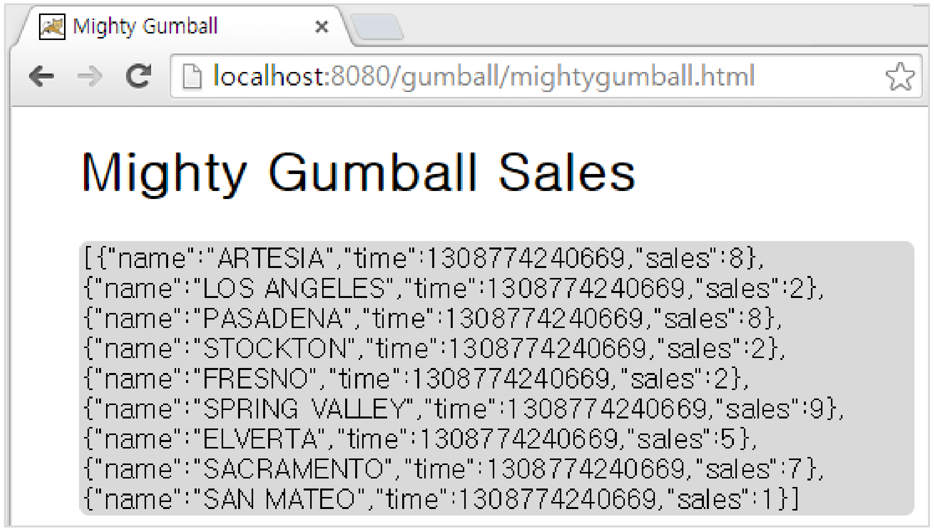
Not pretty, but the data is there.

Nice! That took a lot of work. We had to understand how to do HTTP requests and also set up the server, but it works! I'm already thinking of all the great apps I can build to make use of all the web services out there, now that I know how to talk to them.

Impressing the client…

We've done a lot of heavy lifting to get this app working, and that's great, but Mighty Gumball is going to be a lot more impressed if it looks good too. Here's what we're going for…

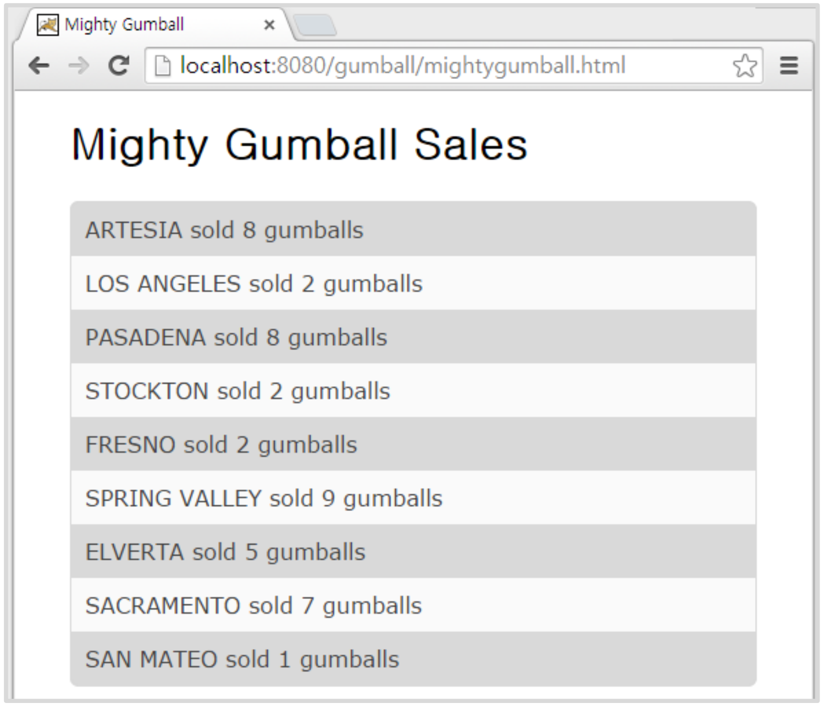
What we have



Mighty Gumball Sales

```
[{"name":"ARTESIA","time":1308774240669,"sales":8},
{"name":"LOS ANGELES","time":1308774240669,"sales":2},
{"name":"PASADENA","time":1308774240669,"sales":8},
{"name":"STOCKTON","time":1308774240669,"sales":2},
{"name":"FRESNO","time":1308774240669,"sales":2},
{"name":"SPRING VALLEY","time":1308774240669,"sales":9},
{"name":"ELVERTA","time":1308774240669,"sales":5},
{"name":"SACRAMENTO","time":1308774240669,"sales":7},
{"name":"SAN MATEO","time":1308774240669,"sales":1}]
```

What we want



Mighty Gumball Sales

ARTESIA sold 8 gumballs
LOS ANGELES sold 2 gumballs
PASADENA sold 8 gumballs
STOCKTON sold 2 gumballs
FRESNO sold 2 gumballs
SPRING VALLEY sold 9 gumballs
ELVERTA sold 5 gumballs
SACRAMENTO sold 7 gumballs
SAN MATEO sold 1 gumballs

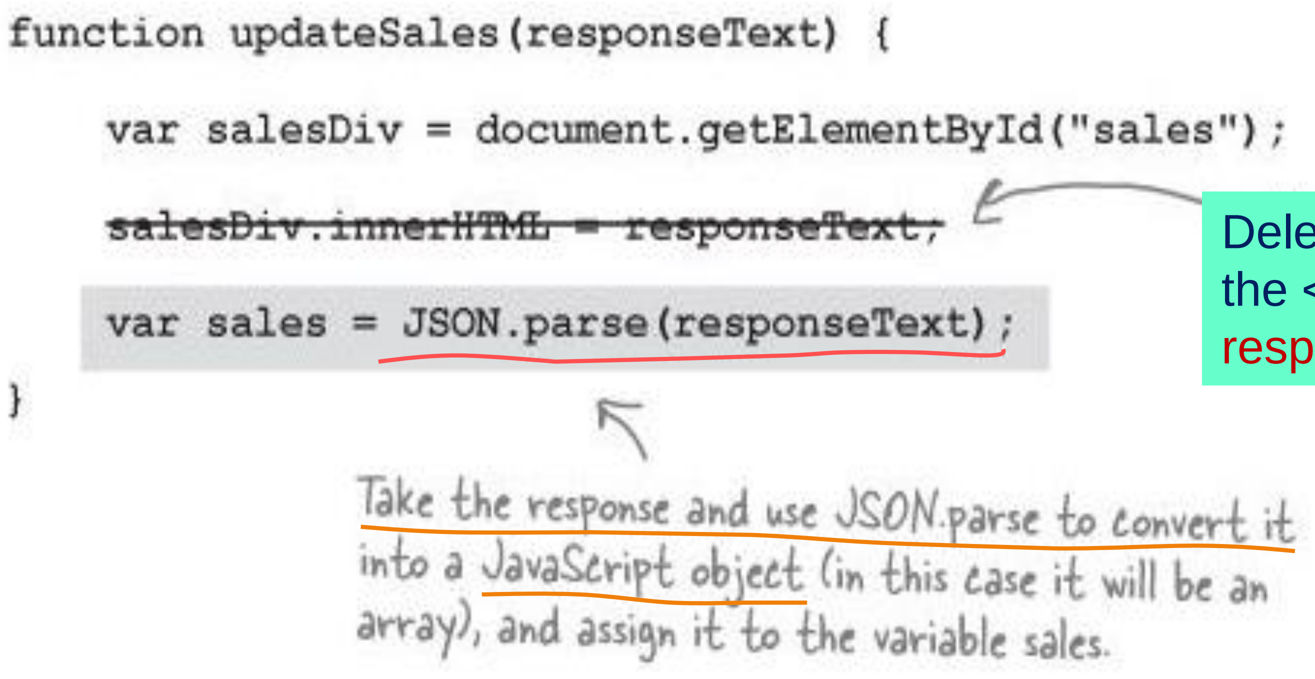
Here's what we need to do to improve our display:

1. Takes the data from our XMLHttpRequest object (JSON string) and converts it into a true JavaScript object.

2. Add new elements to the DOM, one per sales item in the array.

Reworking our code to make use of JSON

1. Convert the **responseText** from a string to its equivalent JavaScript using JSON.parse.

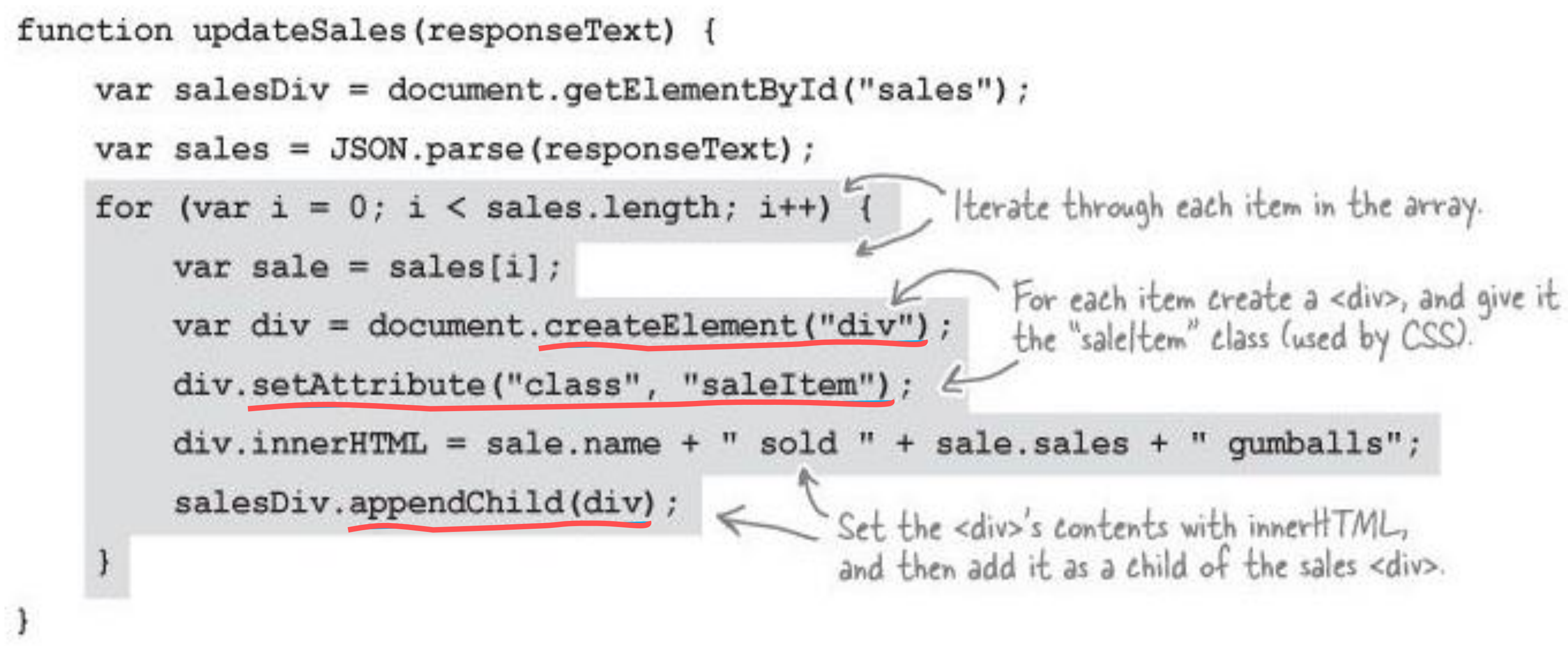


```
function updateSales(responseText) {

    var salesDiv = document.getElementById("sales");

    ~~salesDiv.innerHTML = responseText;~~

    var sales = JSON.parse(responseText);
}
```

Take the response and use JSON.parse to convert it into a JavaScript object (in this case it will be an array), and assign it to the variable sales.

Deleting the line that sets the <div> content to the responseText string

2. **Adds new elements to the DOM**, one per sales item in the array. In this case we are going to create a new <div> for each item:



```
function updateSales(responseText) {
    var salesDiv = document.getElementById("sales");
    var sales = JSON.parse(responseText);
    for (var i = 0; i < sales.length; i++) {
        var sale = sales[i];
        var div = document.createElement("div");
        div.setAttribute("class", "saleItem");
        div.innerHTML = sale.name + " sold " + sale.sales + " gumballs";
        salesDiv.appendChild(div);
    }
}
```

Iterate through each item in the array.

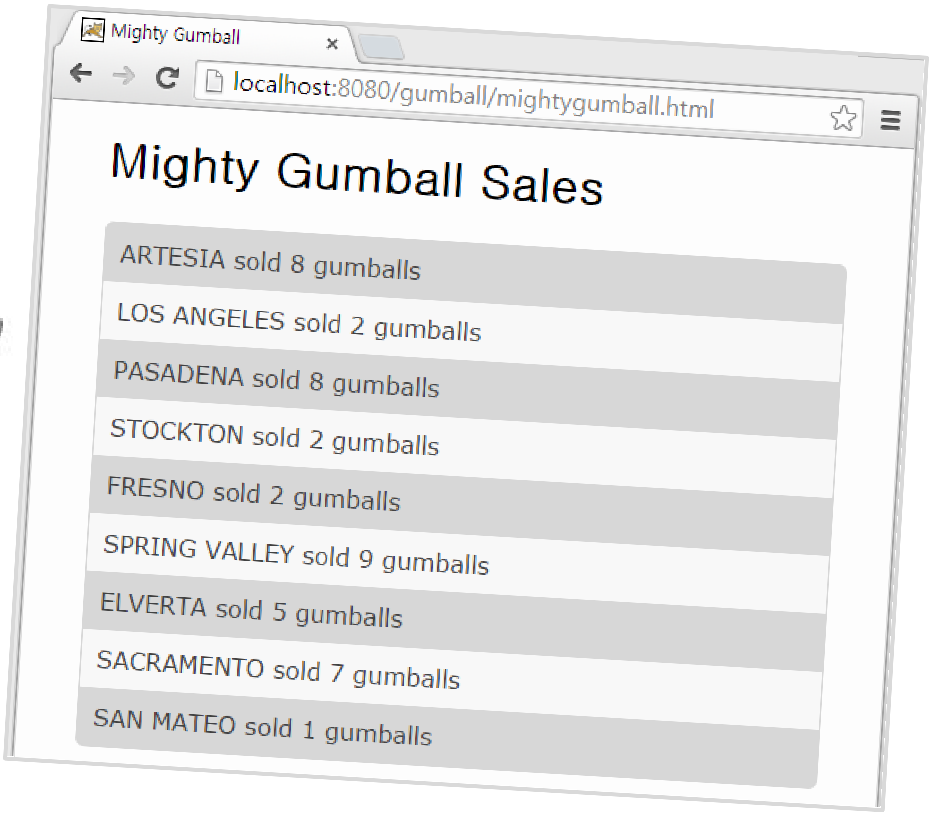
For each item create a <div>, and give it the "saleItem" class (used by CSS).

Set the <div>'s contents with innerHTML, and then add it as a child of the sales <div>.

실습과제 09-2 The Home Stretch...

You already know what this one is going to look like, but go ahead and make these changes. Take one more careful look at the code on the previous page and make sure you've got it all down. Then go ahead, reload that page.

See, we told you it would look like this!



Mighty Gumball Sales

ARTESIA sold 8 gumballs
LOS ANGELES sold 2 gumballs
PASADENA sold 8 gumballs
STOCKTON sold 2 gumballs
FRESNO sold 2 gumballs
SPRING VALLEY sold 9 gumballs
ELVERTA sold 5 gumballs
SACRAMENTO sold 7 gumballs
SAN MATEO sold 1 gumballs

Moving to the Live Server

Testing has gone well, you guys are ready to use Mighty Gumball's live production servers now. Good luck!
Ajay, the Quality Assurance Guy

Mighty Gumball asked us to test locally, and we have. Now we're ready to move on to testing against the real server. This time, rather than retrieving a static JSON data file, we'll be retrieving JSON that is generated dynamically from the Mighty Gumball servers. We do need to update the URL that XMLHttpRequest is using and change it to point to Mighty Gumball. Let's do that:

Here's their server URL. Change this and make sure it's saved.

```
window.onload = function() {
    var url = "http://gumball.wickedlysmart.com";
    var request = new XMLHttpRequest();
    request.open("GET", url);
    request.onload = function() {
        if (request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.send(null);
}
```

실습과제 09-3 A Live Test Drive...

Make sure your URL change is saved in your mightygumball.js file on your server, if you want to keep retrieving your HTML from there, or to your local hard drive if you are using localhost. From there you know what to do: point your browser to your HTML file and watch the live, beautiful, real data from all those people around the world buying Mighty Gumballs!

결과가 나오지 않는다면 그 이유를 설명하시오!!

Q & A