SHAKE THE FUTURE

CENTRALE NANTES

Web Programming

JS and AJAX

JY Martin - JM Normand

Plan

Main objective

In this practical work, we will work on Javascript -JS- and AJAX.

Javascript is a programming language for the web. It is not Java.
AJAX is a Javascript based technology.

The main objective is to understand the main principles of javascript and AJAX and how you can use them.

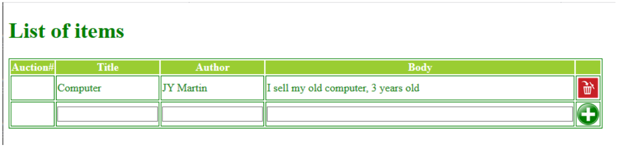
Tools

We need a text editor or an IDE to write HTML/CSS pages.
Same for Javascript.
We will use JQuery, so download it. That will avoid downloading it each time.
For AJAX, we will also need an HTTP server.

Have a look to prerequisites to install tools.

HTML

We will use the HTML/CSS files generated in the 3rd exercice of 01-PRWEB.



List of items

Auction#	Title	Author	Body	
	Computer	JY Martin	I sell my old computer, 3 years old	

Plan

JS basics

There are 2 ways of writing JS scripts.

- Scripts inside an HTML file
- External script files included in HTML files

There are 2 ways of launching scripts.

- Scripts launched by a script tag
- Events based attributes in tags

JS - Writing JS in a HTML file

In your HTML file add this script just after <body>:

```
<script type="text/javascript" >
        document.write("This is my text");
</script>
```

Do not copy/paste these lines, type them in the file. PDF files contains hidden characters that will create errors in your file.

JS - Writing JS in a HTML file

Open your HTML document with a browser.
Text is inserted in your page.

Have a look to the source code from your browser. It is a piece of script.

JS - Writing JS in a HTML file

A bit more. We will split the script.

In the “head” section, add this:

```
<script type=”text/javascript” >
        function writeData() {
                document.write(”This is my text”);
        }
</script>
```

And replace your initial script by this one:

```
<script type=”text/javascript” >
        writeData();
</script>
```

JS - Writing external script files

Scripts do not need to be defined in the HTML file. You can put it in a file and include the javascript file in the HTML file.

Create a file myScript.js
Here is its content.

```
function writeData() {
        document.write("This is my text");
}
```

And replace the script in the head section by this one:

```
<script type="text/javascript" src="myScript.js" ></script>
```

JS - Writing external script files

You can add as many JS files as you want.

Attribute src in the tag “script” is an URL. So you can use any file on internet.

For example, for jquery, you can use:

- http://code.jquery.com/jquery-3.4.1.min.js which will download it from the internet
- if jquery-3.4.1.min.js, is a local file, you refer to the version on your computer.

JS - Scripts launched by a script tag

This is what we already used.

Whatever you place the script tag in the body section, the script will be launched, ... when it is reached in the document.

Try to move the script in the body section of the file, and have a look of what happens.

JS - Use events based attributes

The second way to launch scripts is event based.
That means that when something happens in the page, your browser can catch the event and launch the appropriate action.

For example, you can catch these events:

- page is loaded
- something has change in the page, or in an item (SELECT, INPUT, ...)
- user clicked on an item
- user used keyboard
- ...

JS - Use events based attributes

To catch events, you have to add attributes to the items-s- which have to catch an event.

- onClick to catch a click event
- onChange to catch a modification event
- onKeydown, onKeyup, onKeypressed for the keyboard
- onLoad for the end of document loading
- ...

JS - Use events based attributes

Remove the script in the body section.

Add this attribute to body:

```
<body onkeydown="writeData();" >
```

Relaunch your page. hit any key

Move the onkeydown attribute to an input item. Click in the item an hit a key...

JS - Use events based attributes

How does it work?

An HTML document is a DOM document. When it is read, your browser builds an internal structure to represent the page.

When you add an attribute, you add a new method to the item it is in.

When an event occurs, you browser calls the associate event, if it exists.

What can we do with javascript

... everything. Javascript can manipulate any element in the DOM. It can move elements, remove, update, insert elements, attributes, ...

And as everything is DOM, javascript can do anything.

Display is linked to the DOM, not to the file source. So, anything you change in the DOM leads to a modification on the screen.

Plan

a bit of Javascript syntax

Javascript syntax is not very complicated and is closed to languages like C, C++, Java, ...

One of the big difference is that there is no explicit type in javascript. variable types is chosen according to the immediate use.

javascript is a interpreted language. A file is parsed only if required.

a bit of Javascript syntax

First, variables.

- Like many languages, variables are made of characters. If possible, no accentuation.
- a variable should be defined by:

```
var variable;
```

- Even not defined, you can use a variable
- To set a variable's content:

```
variable = content;
```

a bit of Javascript syntax

Instructions.

- They are ended by ;
- An instructions bloc is enclosed into parenthesis: { ... }

a bit of Javascript syntax

Strings.

- can be used with " or '
- contatenate with +
- integers, doubles, ... are converted to string when it is necessary to.

a bit of Javascript syntax

compare elements.

- ==, !=, <, <=, >, >=
- &&, ||, !
- and other operators
  - === means equals and the same type
  - !== means different or not the same type

a bit of Javascript syntax

Some values you may know:

- null: as many languages null is an object which doesn't exist
- undefined: object value is not defined. Not the same as null

a bit of Javascript syntax

Control structures.

- if (expression) {...}
- if (expression) {...} else {...}
- switch (...) { case ...:... otherwise: ... }
- for (...;...;...) { ... }
- while (...) { ... }

a bit of Javascript syntax

functions:

```
function functionName(parameters) {
        ...
}
```

And if you return a value,

```
function functionName(parameters) {
        ...
        return ...;
}
```

a bit of Javascript syntax

calling functions:

```
functionName(arguments);
```

Or if your return a value:

```
var variable=functionName(arguments);
```

a bit of Javascript syntax

Arrays.

- start at 0
- no size limit
- indexed by integers
- To acces an index: myArray[index]
- myArray.length = array length
- array creation
  - var myArray=[1,2,3]
  - var myArray = new Array()

... and many functions like push, pop, splice, ...

From Javascript to ECMAScript 6

ECMAScript is the true name of Javascript. ECMAScript 6 is the last evolution and introduce some new instructions.

You can have a look to this URL for further informations: http://es6-features.org

That includes:

- new class management.
- using let instead of var for local variable declaration
- variable and function scope modified
- default values for parameters
- ...

Using browser web development tools.

Open you web development tools.

- Firefox: right left icon, “Web development”, “development tools”.
- Chrome: right left icon, “More tools”, “Web development tools”.
- Safari: “Development”,”Web Inspector”.

Using browser web development tools.

Open your HTML page. Then, in your web development tool, select your “myScript.js” file.

Add a breakpoint on the “document.write” line.

Check your browser stop at the breakpoint when you activate the right event.

Using console to log informations

Replace writeData content by this one:

```
function writeData() {
          console.log("This is my text");
}
```

Remove your breakpoint.
Reactivate your event and check the console in the web development tools.
This can be used for debugging scripts too.

Plan

Introduction to DOM

DOM is the **D**ocument **O**bject **M**odel.

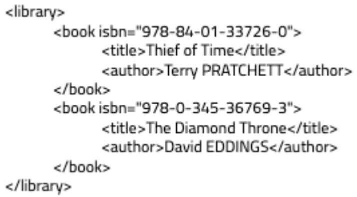
It includes:

- A model for data representation
- A set of method to implement.

Ìt is not a framework, but a set of specifications that have to be implemented.
Javscript, Javax, ... implements DOM specifications.

DOM: abstract memorization model

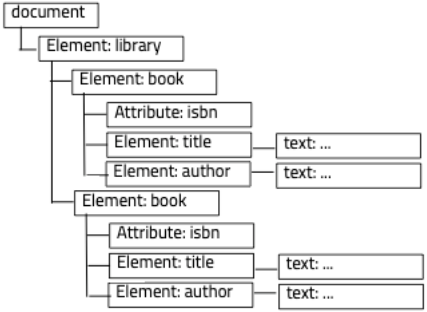
Here is an XML file.



```
<library>
        <book isbn="978-84-01-33726-0">
                <title>Thief of Time</title>
                <author>Terry PRATCHETT</author>
        </book>
        <book isbn="978-0-345-36769-3">
                <title>The Diamond Throne</title>
                <author>David EDDINGS</author>
        </book>
</library>
```

DOM: abstract memorization model

Here is the XML tree representation



DOM: abstract memorization model

Memorization model considers elements as tree nodes.

- document is root
- a text node is a leaf. No son.
- except document, any node knows its parent.
- any node knows document root
- except leaves, any node knows its first child and its last child
- except document, leaves and first nodes, any node knows its previous sibling
- except document, leaves and last nodes, any node knows its next sibling
- ...

DOM: abstract memorization model

for example to browse children of a node, we could do something like this:

```
aChild <- node.firstchild
while (aChild != null) {
        aChild <- aChild.nextSibling
}
```

DOM: node attributes

- tagname: the tag name, except for text nodes
- id: the id, if this attribute has been set
- modeType: tag, text, ... TAG is 1, TEXT is 3.
- parentNode: parent node
- childNodes: array of nodes children
- firstchild: the first child of a node
- lastchild: the last child of a node
- nextsibling: next parent's son
- previoussibling : previous parent's son
- ...

DOM: searching functions

DOM also ask for implementing some methods.

Here are some usefull searching methods:

- getElementById(id): returns node with the given id
- getElementsByTagName(tagName): returns an array of nodes whose tagName is tagName
- getElementsByName(name): returns an array of nodes whose tagName is tagName

These methods are applied on “document”.

DOM: creating, deleting functions

some create, remove methods:

- createElement(tagName): returns a new node with the given tag name. Applied on document.
- createTextNode(text): creates a new text node with the given text. Applied on document.
- appendChild(aNode): link aNode to the current node as its last child
- setAttribute(name, value): add a new attribute to the node with the given name and value.
- removeChild(aNode): remove aNode from children list of the node

and more

DOM: changing properties

Nodes properties can be change by a simple assignment.

For example, setting property to a node:

```
aNode.id = "newID";
```

If the property is an attribut you use points, as usual.

```
aNode.style.display = "none";
```

DOM: creating elements 1/2

Creating elements is done in the document. For example:

```
var myNewTag = document.createElement("INPUT");
myNewTag.setAttribute("type", "text");
myNewTag.setAttribute("name", "theInputName");
myNewTag.setAttribute("value", "theInputValue");
```

Creating an element do not link it to your document. You have to link it manually.

DOM: creating elements 2/2

In this example, we create a TD and a TR and we add the TD at the end of the TR. Then we get the last line of a table with its ID and insert the new TR element just before this last line.

```
var myNewTag = document.createElement("INPUT");
var myTD = document.createElement("TD");
myTD.appendChild(myNewTag);
var myTR = document.createElement("TR");
myTR.appendChild(myTD);
var lastLine = document.getElementById("lastLine");
lastLine.parentNode.insertBefore(myTR, lastLine);
```

DOM: creating text nodes

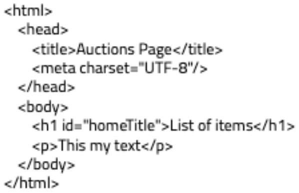
Text nodes are not created the same way as the other nodes. For tags, we use createElement. For text node, there is a specific function to create it: createTextNode.

```
var myNewPar = document.createElement("P");
var myText = document.createTextNode("This is my text");
myNewPar.appendChild(myText);
```

DOM and browsers

Browser uses DOM to represent HTML files.

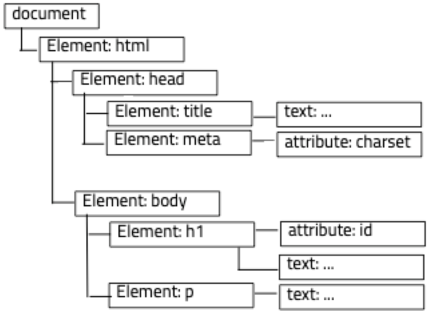
Here is an HTML file:



```
<html>
  <head>
    <title>Auctions Page</title>
    <meta charset="UTF-8"/>
  </head>
  <body>
    <h1 id="homeTitle">List of items</h1>
    <p>This my text</p>
  </body>
</html>
```

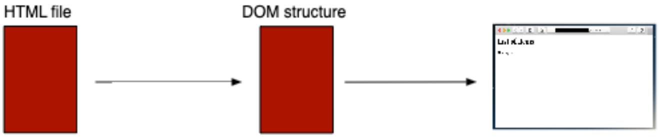
DOM and browsers

And here is the way it is stored in the browser memory.



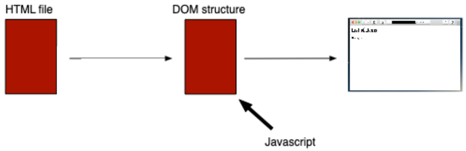
DOM and browsers

And here is the way it works.



DOM and Javascript

Javascript implements DOM functions. So it can manipulate any DOM element.



That means, any element you put in your HTML file, JS can access, delete, modify anything.

And as your browser displays the DOM structure, any modification made by javascript is applied on the view.

Plan

CENTRALE NANTES

Listener and JS function

We will start with a basic action on our HTML file: When we click on the “delete” button, we want the line to be removed.

We will use your “myScript.js” file. Replace writeData by this:

```
function deleteLine() {
        console.log(”delete line”);
}
```

Remove call to writeData in your HTML file.
Add an onclick event listener on your “delete” button. This will call your method deleteLine.
Check message is logged.

Using parameters

Now, we will use parameters.

change your listener in the HTML code by:

```
onclick="deleteLine(this);"
```

And you JS function by:

```
function deleteLine(ref) {
        console.log("delete line" + ref.tagName);
}
```

Add a breakpoint to the log line, click on the delete button. Have a look to "ref".

Getting the line

Ok, now we have to get the TR to get the full line.

2 ways of doing that:

- add an ID to TR
- iterate to get TR from the element.

Getting the line - 1st method 1/1

Add an attribute ID to the TR that contains your delete button.

```
<tr id="myLine" >...
```

And in you script, get the TR tag with:

```
var trRef = document.getElementById("myLine");
```

Reload your HTML file, add a breakpoint in your JS file, and click on the delete button.
trRef should be your TR. Or null if there is a problem.

Getting the line - 2nd method 1/2

We will write a method that goes up to the DOM tree, until it reaches the right tag.

```
function getNextParentTag(currentElement, tagName) {
        // nodeType=1 is TAG
        while ((currentElement !== null)
                        && ((currentElement.nodeType != 1)
                                || (currentElement.tagName !== tagName))) {
                currentElement = currentElement.parentNode;
        }
        return currentElement;
}
```

Getting the line - 2nd method 2/2

Now, we only have to get the TR tag.

```
function deleteLine(ref) {
        var trRef = getNextParentTag(ref, "TR");
}
```

Reload your HTML file, add a breakpoint in your JS file, and click on the delete button.
trRef should be your TR. Or null if there is a problem.

Getting the line - Analysing the 2 methods.

Both of the methods will gice you the result you want, bu not with the seame constraints.

- 1st method requires an id for each line.
  For example, as we manage auctions, first column is the auction id. So we can use this for our IDs and calls
  - let id be the auction id
  - TR id could be myTR_id
  - use id as an argument when you call deleteLine
    deleteLine(this, id)
  - rebuild TR id in deleteLine(ref, theID)
    var myTRTag = "myTR" + theID;
- 2nd method requires the HTML hierarchy to be well formed.

Deleting line

We've got the TR tag. We only have to remove it from document.

Try with:

```
function deleteLine(ref) {
        var trRef = getNextParentTag(ref, "TR");
        if (trRef !== null) trRef.parentNode.removeChild(trRef);
}
```

Reload your HTML file, and click on the delete button.

Modifying items

We would like to be able to modify the line.

Add a new Edit button on the line, just beside the delete button.
Add a onclick listener that calls an editLine function.
Create an editLine function in your JS file.

Use the method you want to get the TR of the line in the editLine function.

Modifying items

Create a function deleteAll that, according to a node, remove its content.

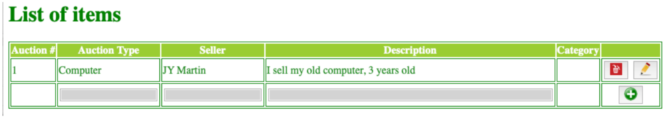
You can use the fact that, if you remove the firstChild of a node, then, the second one will be the firstChild.
If there is not other child, firstChild is set to null.

Create a function which, according to a TR node, get successively the TD nodes.
For TD author, body, title, and category, remove content. Keep the id in first TD.

Reload your HTML file, and click on the edit button.

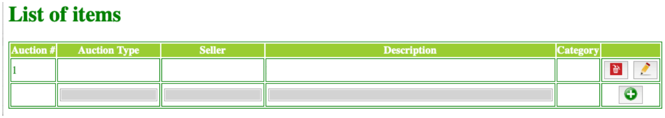
Modifying items

You should have something like that:



List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JY Martin	I sell my old computer, 3 years old		

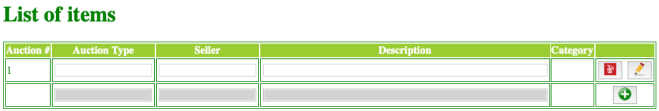


List of items

Auction #	Auction Type	Seller	Description	Category	
1					

Modifying items

Modify your function so that, for author, body and title, content is replaced by an input.


List of items

Auction #	Auction Type	Seller	Description	Category	
1					

Modifying items

Add a new Save button on the line, just beside the edit button.
Set it as invisible using style="display:none;"
Force delete and edit button to be visible with
style="display:inline;"

When you have switch content to input tags, you can hide edit button and show save button.
Remember button.style.display="inline;" will show the item. none will hide it.
We could have use block instead of inline but it would display buttons on different lines. Inline will keep them on the same line.

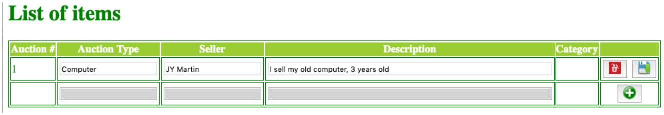
Maybe you can first show all buttons, then hide the calling one...

Saving items

Add a listener in your save button and scripts in your JS file so that when you click on save, inputs are removed, save button is hidden and edit button is shown.

Reload your HTML file, and click on the edit button, the save, edit, ...

... if you are motivated enough, take data from the inputs and display it as text.



List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JY Martin	I sell my old computer, 3 years old		

Adding elements 1/2

Now, we want to take into account the "add" button.

Add an addLine function to your JS file. The button will be your parameter, as we did for deleteLine.

To get all input tags in the page, you can use something like:

```
var myArray = document.getElementsByTagName("INPUT");
```

or if you want to get each input, you can use using something like this:

```
var myArray = document.getElementsByName("author");
```

Adding elements 2/2

For each input element, get its value. Maybe you could create a JSON object for that.

Then create a new line and as many TD tags as needed. Add text elements according to the contents it may have.

If you skipped it, maybe you should have a look to DOM and JSON chapters.

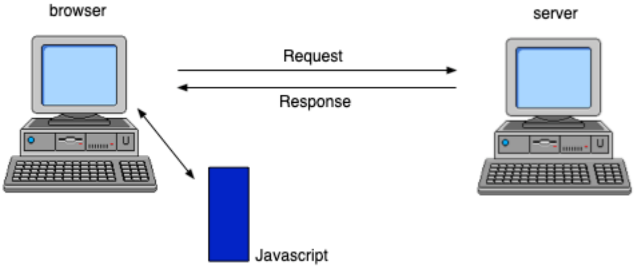
Check all

Check all your buttons are operational.

Plan

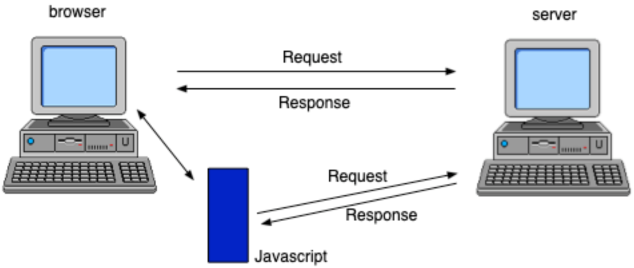
AJAX principles

Current exchanges between a browser and a client could look like this:



AJAX principles

But could javascript directly exchange with the server?



AJAX principles

We have to take into account some constraints:

- Server is listening for an HTTP request.
- HTTP request are GET or POST request.
- For security reasons, javascript cannot send a request to any server.
- Javascript do not need an HTML response, only data to modify the current page.

Javascript/AJAX requests hace to respect the HTTP protocol.
A specific object is created for that: xmlHttpRequest.

AJAX principles

Javascript do not need an HTML response, only data to modify the current page.

For an AJAX request, response is not an HTML page but a data set. 3 formats are defined:

- text mode
- XML mode
- JSON mode

AJAX: how does it work?

- a javascript function is called. It requires data to complete.
- the javascript function send a request to the server
- server process the request and replies
- the javascript function gets the response and get requested data
- the javascript function uses data to alter the current page.

AJAX: synchronous and asynchronous

When the javascript function calls the server, it as to wait for an answer.

There are 2 way of doing that:

- Stop function and wait until the response is sent.
  This is the **synchronous** way
- Launch a background function -a callback function- that will be in charge of waiting for the response.
  When response is sent, the callback function gets it and process the response.
  This is the **asynchronous** way

AJAX: synchronous and asynchronous

What does it change?

- During a synchronous call, javascript function is locked. it waits for the answer.
- Using an asynchronous call lets the process running in background. You can cancel call if necessary, ...

A bit of JSON

Java**S**cript **O**bject **N**otation

This is a data format can be used to represent a set of data.

As they are designed for javascript, they are fully adapted to coding/decoding for javascript.
Javascript includes a library that can encode, decode data: json.

A bit of JSON

Here are the object you can manipulate:

- an object: **{ ... }**
- a field: **“fieldName”: value**
- an array: **[...]**

values can be integers, strings, reals, objects, arrays, ...
json objects are lists of fields and arrays.
json arrays are arrays of objects

field elements and arrays elements are separated by a comma.

A bit of JSON

Here is an example of a JSON object:

```
{
        "firstname": "myFirstName",
        "lastname": "myLastName",
        "diploma": [
                "Master Computer Science",
                {
                        "name": "PhD Computer Science",
                        "year": "1990"
                }
        ]
}
```

A bit of JSON

How can we create a JSON object? Using JSON functions.

```
var str = '{ "firstname": "myFirstName", "lastname": "myLastName" }';
var myJsonObject = JSON.parse(str);
```

fields names and fields values **MUST** be into quotation marks.
JSON.parse uses a json **object** as a parameter, never a field nor an array.

AJAX and JQuery

JQUery is one of the most famous javascript library, evean if son libraries like ANgularJS and React are replacing JQuery functions.

JQUery implements many functions, including **ajax** function. We will use this one.
This function has a JSON parameter that include at least:

- called url
- data to send
- a success function that handles the response

AJAX and JQuery

Here is a basic call:

```
$.ajax( {
        url:"processAJAX.php",
        data: {
                "name": "myName",
                "choice": 1
                },
        success: function(result) {
                document.getElementById("res").value = result.data;
        }
});
```

AJAX and JQuery

Here are the more often used parameters:

- url: the URL to call. Required.
- data: JSON object that contains data to send. Required.
- method: method to use. Use GET (default) or POST.
- async: asynchronous or not. Use true or false, true is default value.
- success: function to call if result is successful. Parameter is a JSON object. Required.
- error: function to call if result is an error. Parameters are
  - result: data send by the server
  - textStatus: error sent by the server
  - errorThrown: error sent by JQuery

AJAX and JQuery

Some remarks:

- Even if it is not necessary, we recommand using parameters "method" and "error".
- Remember the function is **asynchronous** by default. That means functions "success" or "error" will be called when the response is catched.
  "ajax" function ends and instructions below "ajax" are executed while "success" or "error" are still waiting.

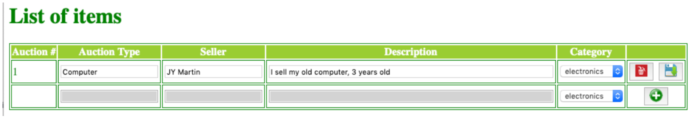
Let’s use AJAX

- Add jquery to the javascript includes of your HTML file.
- Add file “action.php” in the materials to your server files.
- When the “add” button is clicked, catch “author”, “body”, “title” and send them to “action.php”.
- You should have a JSON response with “id”, “author”, “title”, “body” and “categoryName”.
  Use them to display the added line.

Use categories

And if you think you can do it, add categories:

- Display them when you edit a line
- Display them when you add an item
- Take them into account in your AJAX calls
- action.php can handle categories.



List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JY Martin	I sell my old computer, 3 years old	electronics	
				electronics	

Success
NEXT EXIT