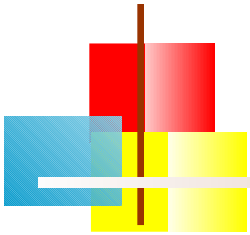
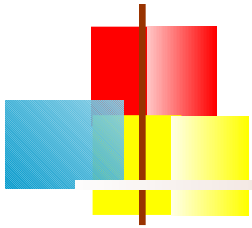


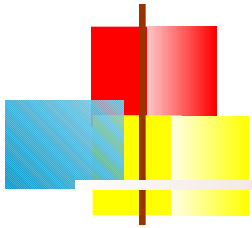
Ajax

David Matuszek's presentation,
http://www.cis.upenn.edu/~matuszek/cit597-2007/index.html



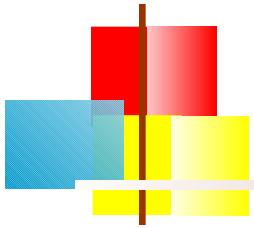
The hype

- Ajax (sometimes capitalized as AJAX) stands for **A**synchronous **J**avaScript **A**nd **X**ML
- Ajax is a technique for creating “better, faster, more responsive web applications”
- Web applications with Ajax are supposed to replace all our traditional desktop applications
- These changes are so sweeping that the Ajax-enabled web is sometimes know as “Web 2.0”



The reality

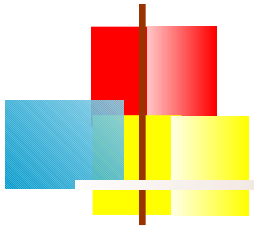
- Ajax ***is*** a technique for creating "better, faster, more responsive web applications"
  - But they still aren't as responsive as desktop applications, and probably never will be
  - Web applications are useless when you aren't on the web
- GUIs are HTML forms (and you know how beautiful and flexible those are)
- The technology has been available for some time
- AJAX is not a new programming language, but a new way to use existing standards.
  - Google uses it extensively, in things like Google Earth and Google Suggest
  - It has been given a catchy name
- Ajax ***is*** a useful technology, and a good thing to have on your resumé



How Ajax works

- You do something with an HTML form in your browser
- JavaScript on the HTML page sends an HTTP request to the server
- The server responds with a *small amount* of data, rather than a complete web page
- JavaScript uses this data to modify the page

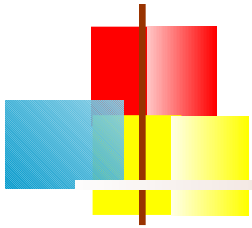
- This is faster because less data is transmitted and because the browser has less work to do



Underlying technologies

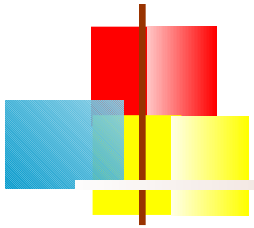
- JavaScript
- HTML
- CSS
- XML
  - XML is often used for receiving data from the server
  - Plain text can also be used, so XML is optional
- HTTP

- All these are open standards



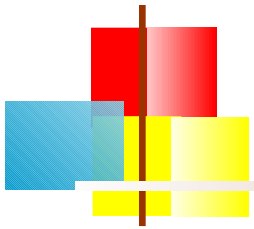
Starting from the browser…

- Your browser *must* allow JavaScript, or Ajax won't work
  - Otherwise, there's nothing special required of the browser
- Your browser has some some way of providing data to the server—usually from an HTML form
- JavaScript has to handle events from the form, create an XMLHttpRequest object, and send it (via HTTP) to the server
  - Nothing special is required of the server—every web server can handle HTTP requests
  - Despite the name, the XMLHttpRequest object does not require XML



The XMLHttpRequest object

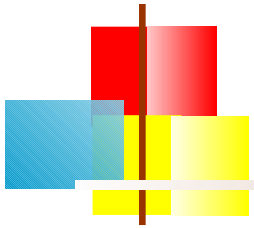
- JavaScript has to create an XMLHttpRequest object
- For historical reasons, there are three ways of doing this
  - For most browsers, just do
    `var request = new XMLHttpRequest();`
  - For some versions of Internet Explorer, do
    `var request = new ActiveXObject("Microsoft.XMLHTTP");`
  - For other versions of Internet Explorer, do
    `var request = new ActiveXObject("Msxml2.XMLHTTP");`
- Doing it incorrectly will cause an Exception
- The next slide shows a JavaScript function for choosing the right way to create an XMLHttpRequest object



Getting the XMLHttpRequest object

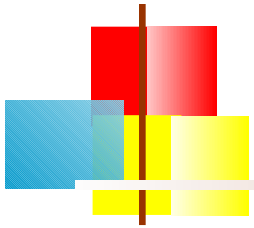
```
var request = null;    // we want this to be global

function getXMLHttpRequest( ) {
    try { request = new XMLHttpRequest(); }
    catch(err1) {
        try { request = new ActiveXObject("Microsoft.XMLHTTP"); }
        catch(err2) {
            try { request = new ActiveXObject("Msxml2.XMLHTTP"); }
            catch(err3) {
                request = null;
            }
        }
    }
    if (request == null) alert("Error creating request object!");
}
```



Preparing the XMLHttpRequest object

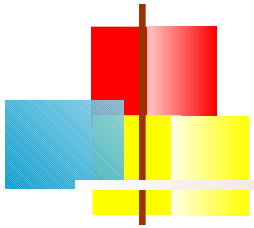
- Once you have an XMLHttpRequest object (***request***), you have to prepare it with the open method
- ***request***.open(***method***, ***URL***, ***asynchronous***)
  - The ***method*** is usually 'GET' or 'POST'
  - The ***URL*** is where you are sending the data
    - If using a 'GET', append the data to the URL
    - If using a 'POST', add the data in a later step
  - If ***asynchronous*** is true, the browser does not wait for a response (this is what you usually want)
- ***request***.open(***method***, ***URL***)
  - As above, with ***asynchronous*** defaulting to true



Sending the XMLHttpRequest object

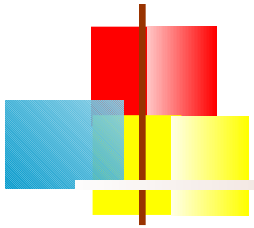
- Once the XMLHttpRequest object has been prepared, you have to send it
- *request*.send(null);
  - This is the version you use with a GET request
- *request*.send(*content*);
  - This is the version you use with a POST request
  - The content has the same syntax as the suffix to a GET request
  - POST requests are used less frequently than GET requests
  - For POST, you *must* set the content type:

```
request.setRequestHeader('Content-Type',
                         'application/x-www-form-urlencoded');
request.send('var1=' + value1 + '&var2=' + value2);
```



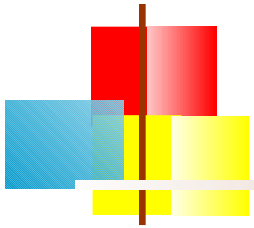
The escape method

- In the previous slide we constructed our parameter list to the server
  - request.send('var1=' + *value1* + '&var2=' + *value2*);
- This list will be appended to the URL (for a GET)
- However, some characters are not legal in a URL
  - For example, spaces should be replaced by %20
- The escape method does these replacements for you
  - request.send('var1=' + escape(*value1*) + '&var2=' + escape(*value2*));



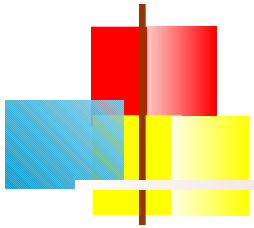
Putting it all together

- Head:
  - function getXMLHttpRequest () { ... } // from an earlier slide
  - function sendRequest() {
    getXMLHttpRequest();
    var url = ***some URL***
    request.open("GET", url, true); // or POST
    request.onreadystatechange = handleTheResponse;
    request.send(null); // or send(***content***), if POST
  - function handleTheResponse() {
    if (request.readyState == 4) {
    if (request.status == 200) {
    var response = request.responseText;
    // do something with the response string
    } else {
    alert("Error! Request status is " + request.status);
    }
    }
    }
- Body: <input value="Click Me" type="button" onclick="sendRequest">



On the server side

- The server gets a completely standard HTTP request
- In a servlet, this would go to a doGet or doPost method
- The response is a completely standard HTTP response, but…
- …Instead of returning a complete HTML page as a response, the server returns an arbitrary text string (possibly XML, possibly something else)

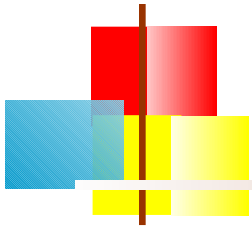


Getting the response

- Ajax uses asynchronous calls—you don't wait for the response
- Instead, you have to handle an event
  - request.onreadystatechange = someFunction;
    - This is a function assignment, *not* a function call
      - Hence, there are ***no parentheses*** after the function name
    - When the function is called, it will be called with no parameters
  - function someFunction() {

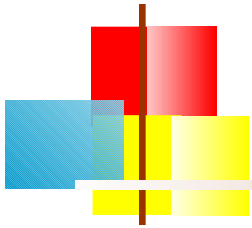
```
    if(request.readyState == 4){
        var response = request.responseText;
            if (http_request.status == 200) {
                // Do something with the response
            }
    }
}
```

- To be safe, set up the handler *before* you call the send function



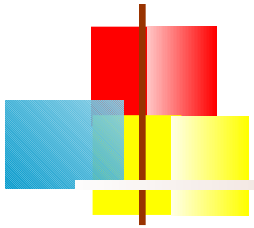
The magic number 4

- The callback function you supply is called not just once, but (usually) four times
- request.readystate tells you why it is being called
- Here are the states:
  - 0 -- The connection is uninitialized
    - This is the state before you make the request, so your callback function should not actually see this number
  - 1 -- The connection has been initialized
  - 2 -- The request has been sent, and the server is (presumably) working on it
  - 3 -- The client is receiving the data
  - 4 -- The data has been received and is ready for use
- I don't know any reason ever to care about the other states
  - I guess the browser just wants you to know it's not loading



The magic number 200

- A ready state of 4 tells you that you got a response--it doesn't tell you whether it was a *good* response
- The http_request.status tells you what the server thought of your request
  - 404 Not found is a status we are all familiar with
  - 200 OK is the response we hope to get



Using response data

- When you specify the callback function,

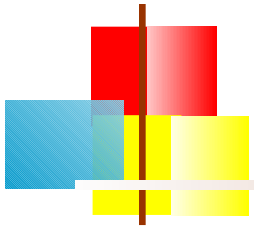
```
request.onreadystatechange = someFunction;
```

  you can't specify arguments

- Two solutions:
  - Your function can use the request object as a global variable
    - This is a very bad idea if you have multiple simultaneous requests
  - You can assign an anonymous function:

```
request.onreadystatechange = function() { someFunction(request); }
```

    - Here the anonymous function calls your someFunction with the request object as an argument.

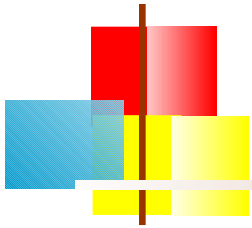


Displaying the response

```
http_request.onreadystatechange =
      function() { showContentsAsAlert(http_request); };
http_request.open('GET', url, true);
http_request.send(null);
```

```
function showContentsAsAlert(http_request) {
    if (http_request.readyState == 4) { /* 4 means got the response */
        if (http_request.status == 200) {
            alert(http_request.responseText);
        } else {
            alert('There was a problem with the request.');
        }
    }
}
```

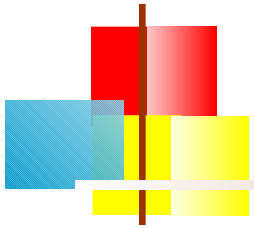
From: http://developer.mozilla.org/en/docs/AJAX:Getting_Started



The readyState property

- The readyState property defines the current state of the XMLHttpRequest object.
- Here are the possible values for the readyState property:
  - **readyState=0** after you have created the XMLHttpRequest object, but before you have called the open() method.
  - **readyState=1** after you have called the open() method, but before you have called send().
  - **readyState=2** after you have called send().
  - **readyState=3** after the browser has established a communication with the server, but before the server has completed the response.
  - **readyState=4** after the request has been completed, and the response data have been completely received from the server.
- Not all browsers use all states
- Usually you are only interested in state 4

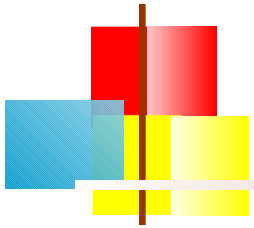
Mostly from: http://www.w3schools.com/ajax/ajax_xmlhttprequest.asp



Summary

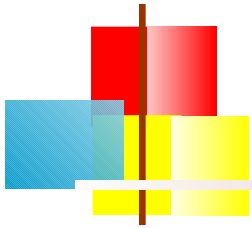
- Create an XMLHttpRequest object (call it *request*)
- Build a suitable URL, with ?*var*=*value* suffix
- *request*.open('GET', *URL*)
- *request*.onreadystatechange = *handlerMethod*;
- *request*.send(null);
- function *handlerMethod*() {

```
    if (request.readyState == 4) {
        if (http_request.status == 200) {
            // do stuff
        }
    }
}
```



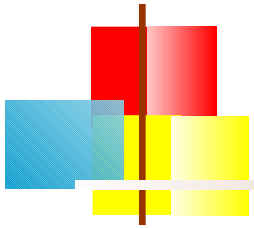
Back to the HTML DOM

- Once we get a response back from the server, we probably want to update our HTML page
- The HTML page itself is document
- You can get information from the HTML page
  - var price = document.getElementById("price").value;
  - var allImages = document.getElementsByTagName("img");
  - var firstImg = document.getElementsByTagName("img")[0];
- We can use the **DOM** to change the HTML :-)



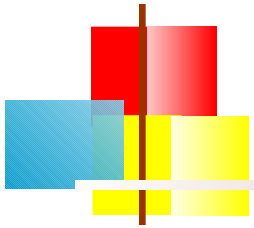
Adding and removing event handlers

- You can add event handlers to HTML elements
  - `<input type="Submit" value="Submit" onClick="doIt();" />`
  - `<img src="dave.jpg" alt="me" onClick="sendMail();" />`
- You can also add handlers programmatically, from a JavaScript function:
  - ```
    var act = document.getElementById("act");
    act.onclick = takeAction;
    ```
  - ```
    var images = document.getElementsByTagName("img");
    for (var i = 0; i < images.length; i++) {
        images[i].onclick = expandImage;
    }
    ```
    - Inside expandImage, the particular image is in the variable `this`
  - Remember: JavaScript is case sensitive, HTML isn't!
- You can programmatically remove event handlers
  - `act.onclick = null`



and

- ... and ... are containers
  - Like <p> for paragraph, there is a blank line before and after a
  - Like <i> for italics, does not affect spacing or flow
- The primary use of these tags is to hold an id attribute
- With an id, we can manipulate page content
  - ```
    // Find thing to be replaced
    var mainDiv = document.getElementById("main-page");
    var orderForm = document.getElementById("target");
    ```
  - ```
    // Create replacement
    var paragraph = document.createElement("p");
    var text = document.createTextNode("Here is the new text.");
    paragraph.appendChild(text);
    ```
  - ```
    // Do the replacement
    mainDiv.replaceChild(paragraph, target);
    ```
    ```



innerHTML

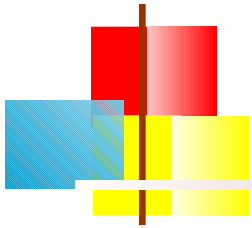
- innerHTML is a non-W3C DOM property that gets or sets the text between start and end tags of an HTML element
  - When the innerHTML property is set, the given string completely replaces the existing content of the object
  - If the string contains HTML tags, the string is parsed and formatted as it is placed into the document
- Syntax:

  var ***markup*** = element.innerHTML;

  element.innerHTML = ***markup***;
- Example:

  document.getElementById(***someId***).innerHTML = ***response***;

- innerHTML is nonstandard, unreliable, and deprecated



The End