Universidad
Politécnica
de Cartagena

Applications on the Internet

Lesson 4. Operation of the dynamic Web

Learning outcomes

- *Recognize in an HTML document the function of the different elements of a form*
- *Identify HTTP methods for sending user information.*
- *Distinguish the HTTP server from the application that processes the user information.*
- *Explain the interaction between an HTTP server and external applications and the communication procedures between the two.*
- *Process information and generate dynamic content using the PHP language.*
- *Explain the need to maintain state information for application development.*
- *Describe the format and attributes of cookies*
- *Describe the management of cookies on the client*
- *Develop applications that store data persistently and manage it efficiently.*
- *Enumerate events of the browser.*
- *Develop interactive applications with javascript, invoking functions as a result of events.*
- *Modify the contents of an HTML document using javascript*
- *Apply mechanisms to asynchronously update a document with AJAX.*

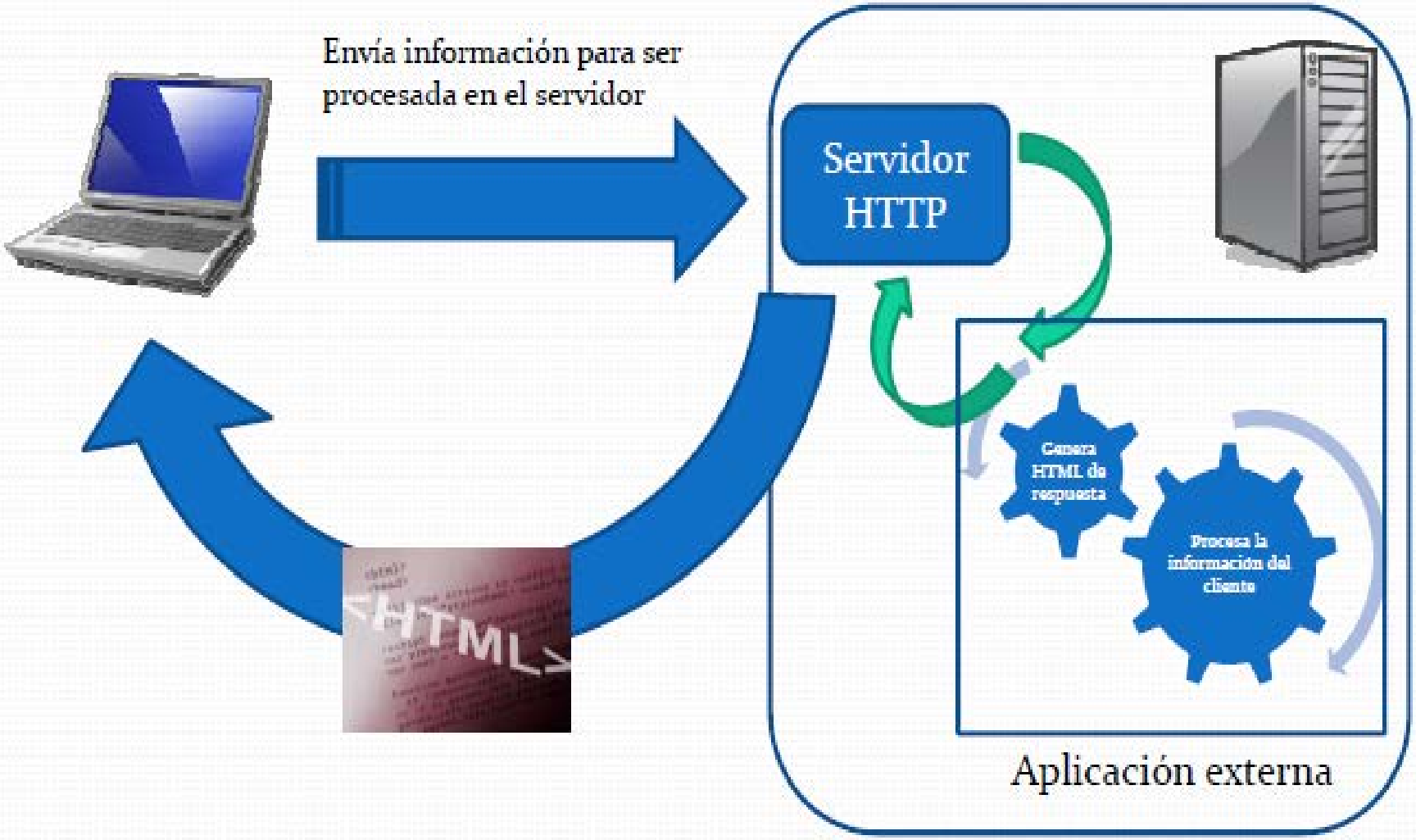
contents

Introduction

What is the dynamic Web?

- As the Web evolves, served content is no longer static (HTML documents linked to each other).
- The **content is generated dynamically** by **program for each request.**
- Web resources that change depending on many factors, namely:
  - Information provided by the user.
  - In response to user actions.
- The **servers process the additional information** that comes **from the client** and generate a response, usually in HTML format, as well as CSS and executable code on the client.

Introduction



Introduction

How a more dynamic Web is achieved? What mechanisms are introduced?

- **Applications on the server side**: Process requests and user information and generate the answer.
- **Code execution on the client**: Allow user interactivity and richer and more attractive interfaces.
- Extensive use of **structured data storage**: Databases and other specialized sources.
- **Data exchange between machines**

contents

Information processing on the server

Information processing on the server side allows the deployment of applications on the infrastructure of the Web.

Advantages over "desktop“ applications:

- Ubiquitous access.
- Multiplatform support: only a compatible browser is needed.
- Code optimization for the server platform.
- Centralized maintenance and upgrades.
- Code integrity and flexibility.

Disadvantages over "desktop“ applications:

- Dependence on the state and quality of the network.
- Less flexibility in the user interface
- Limited capacity (for security) to use the resources of the client computer.
- Data persistence, and more complex state.
- Scalability.

Solve problems

Information processing on the server requires to provide mechanisms to address at least the following issues:

- How the information is sent to the server?
- How the information is processed on the server?
- Who processes it?
- How persistent data and status information is provided to the server?

Sending information to the server

1: What is processed? Indicate the information

We need that the HTML document tells the user the information needed by the server and allows the user to enter it.

- **HTML forms**: Element <form> and elements <input>, <select> <textarea>, etc.

2: Who processes the information? Indicate the destination of the information.

- The **form always contains the URL** of the application that will process the information. With the attribute *action= "URL"* of the <form> element.

3: How is it sent? Indicate the operation of HTTP to use

- The information is sent via HTTP. But **it is done differently depending on the HTTP method**.
- The method is always specified using the attribute *method= "GET | POST "* of the <form> element.

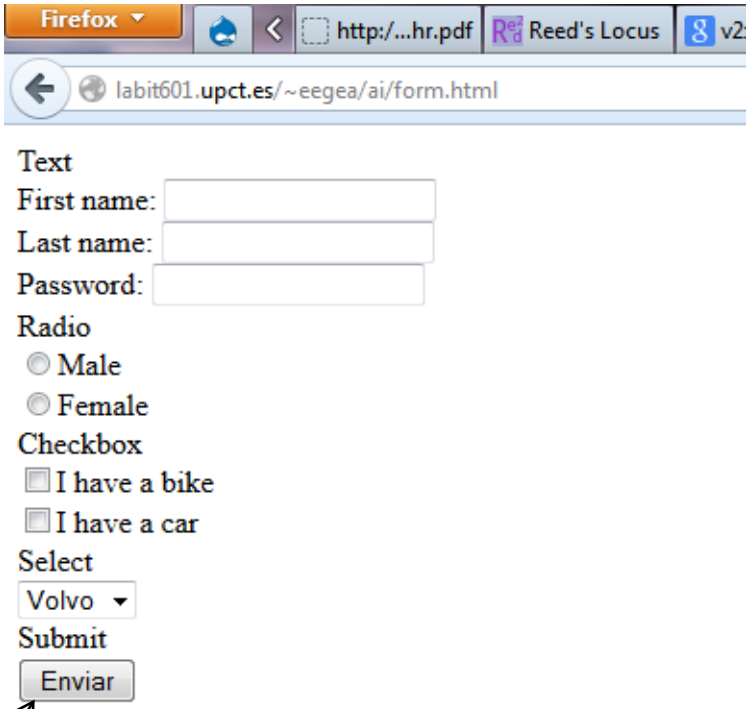
HTML forms

URL of the application that will process the data

Submitting method

```
<!DOCTYPE html>
<html>
<body>

<form action="procesar.php" method="GET">
Text
<br/>
First name: <input type="text" name="firstname">
<br/>
Last name: <input type="text" name="lastname">
<br>
Password: <input type="password" name="pwd">
<br/>
Radio
<br/>
<input type="radio" name="sex" value="male">Male<br>
<input type="radio" name="sex" value="female">Female
<br/>
Checkbox
<br/>
<input type="checkbox" name="vehicle" value="Bike">I have a
<input type="checkbox" name="vehicle" value="Car">I have a
<br/>
Select
<br/>
<select name="cars">
<option value="volvo" selected>Volvo</option>
<option value="saab">Saab</option>
</select>
<br/>
Submit
<br/>
<input type="submit" value="Enviar">
</form>

</body>
```

Variable name



Submit button

HTML forms

- All the fields both <input> as well as <select> or <textarea> must have the attribute ***name*** indicating the **variable name** which will be sent to the server.
- The **value of the variable** corresponds to the information entered by the user.
- Even though the submit button does not appear, the browser may send the information when enter is pressed. It is called implicit submission but does not work always
- There is the <input type= "hidden"> that is ***not shown to the user*** but sends the value of the variable to the server. Example: <input type= "hidden" name= “id" value= "15">
- You can insert other elements between the fields of a form, such as images or other.

Submitting methods

The information is sent to the server using the HTTP protocol.

Using the GET or POST HTTP method results in different ways of sending the information

- GET requests: user information is encoded in the URL of the request.
- POST requests: user information is included in the HTTP request packet.

Files can only be sent to the **server using POST type requests.**

The browser constructs the appropriate request when clicking the submit button.

GET requests

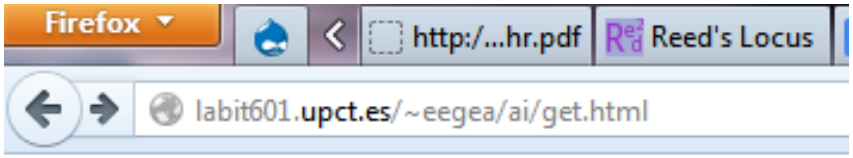
User information is encoded in the URL of the request.

- A *query string* is added to the URL that will process the form.
- Pairs of *variable* = *value* separated by *&*

- http://ait.upct.es/procesar.php?name1=value1&name2=value2&name3=value3

- Special characters are encoded according to URL encoding (RFC 3986):
  - For example, spaces are translated into "+"
  - Percent encoding:%NNN (NNN: number of extended ASCII code, for example).

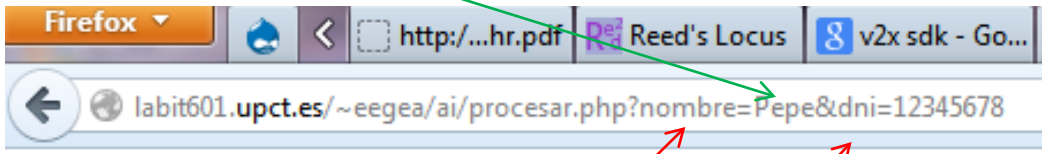
GET requests



Nombre: Pepe

DNI: 12345678

Enviar



Hola Pepe, tu DNI es 12345678

```
<html>
<body>

<form action="procesar.php" method="GET">
<br/>
Nombre: <input type="text" name="nombre">
<br/>
DNI: <input type="text" maxlength="8" name="dni">
<br>
<input type="submit" value="Enviar">
</form>
```

POST requests

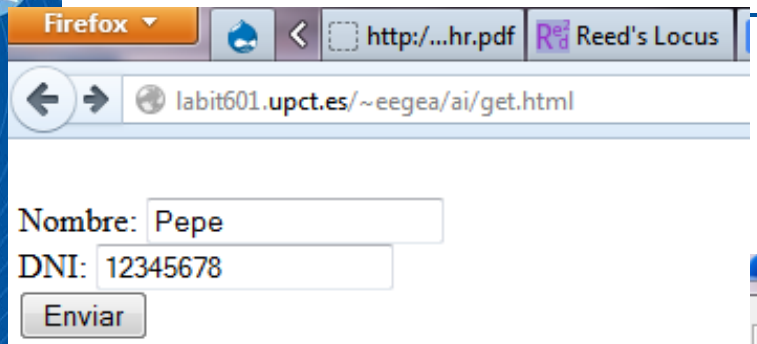
The parameters are sent in the HTTP packet, after the request line and headers.

- Adds nothing to the URL of the request.
- Special characters are encoded by their ASCII extensions.

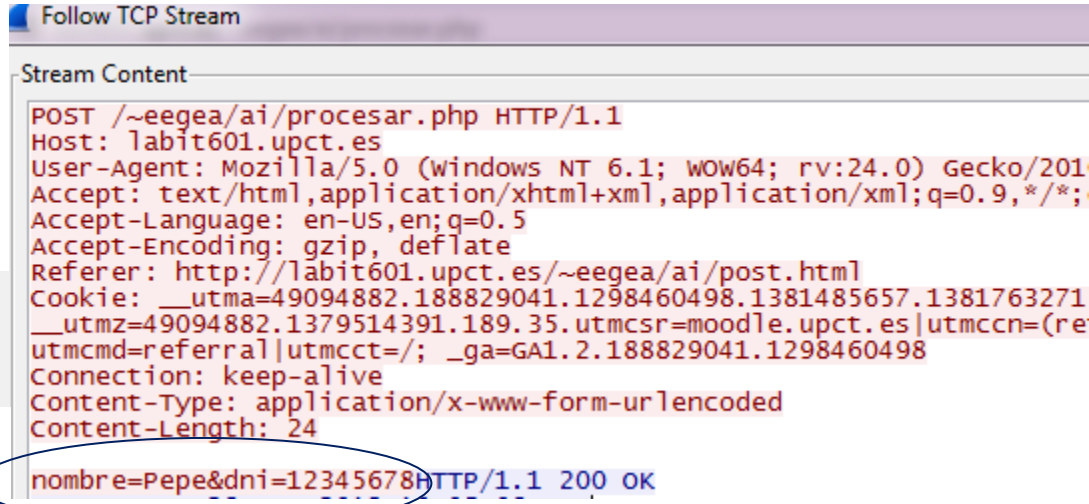
Coding method is indicated in the attribute *enctype* of the <form>

- **application/ x-www-form-urlencoded**: default. They are encoded as **query string** but within the HTTP packet.
  - **Cannot** send **files**
- **multipart/form-data:** a random mark (*boundary*) is used to separate the parameters
  - **It allows to** submit **files**
  - The boundary has to follow certain rules:
    - Start with two dashes --
    - 70 characters (ASCII) maximum
    - End boundary is the same but with just two ending --
  - The variables are sent between boundaries, including the name and *content-type* of the field if necessary.

POST requests - Default

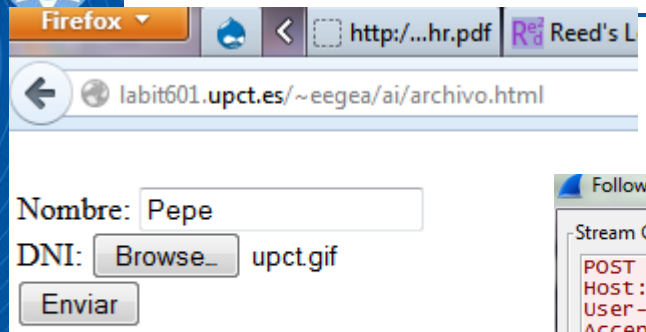


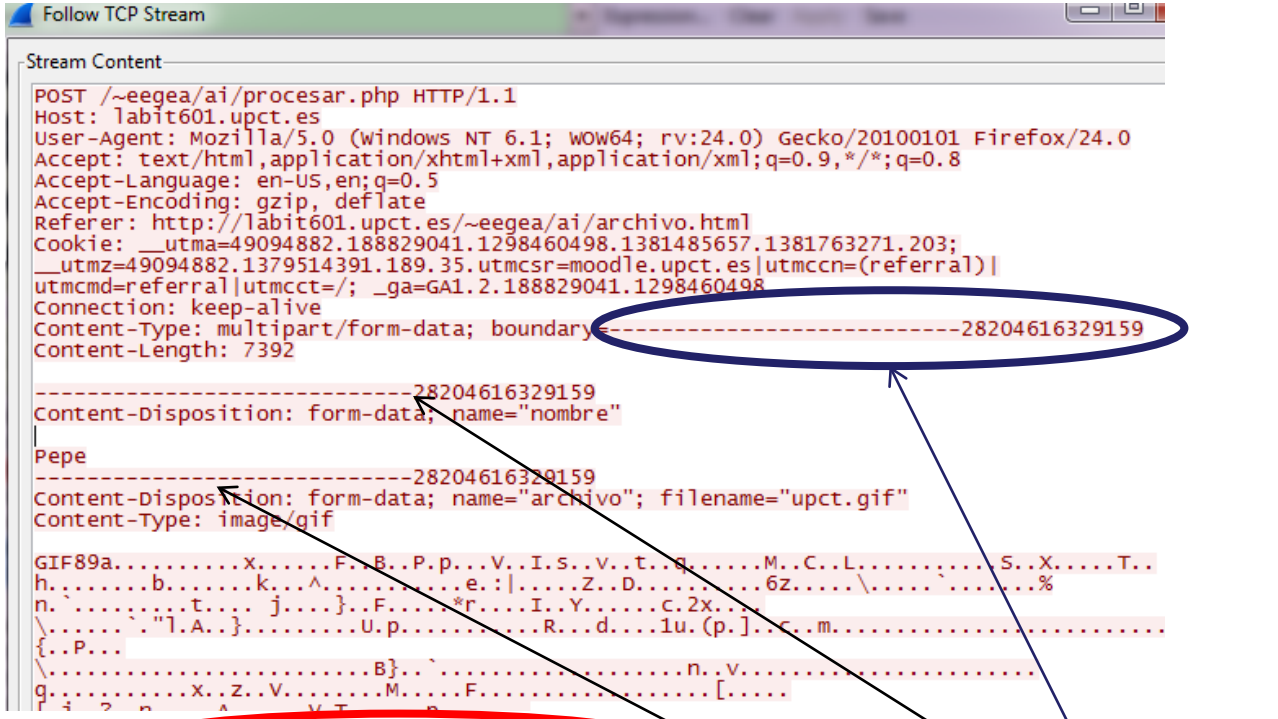

```
<html>
<body>

<form action="procesar.php" method="POST">
<br/>
Nombre: <input type="text" name="nombre">
<br/>
DNI: <input type="text" maxlength="8" name="dni">
<br>
<input type="submit" value="Enviar">
</form>
```



Information encoded in the form of query string in the HTTP packet

POST requests - multipart






```
<form action="procesar.php" method="POST" enctype="multipart/form-data">
<br/>
Nombre: <input type="text" name="nombre">
<br/>
DNI: <input type="file"  name="archivo">
<br>
<input type="submit" value="Enviar">
</form>
```

Attribute enctype required for sending files

¿GET or POST?

HTTP specification indicates that

- GET means "get" information. **Idempotent operations**. Example: search users.
- POST involves “doing" an action with a "side effect". **Non-idempotent operations.** Example: registering a new user.

Some additional considerations:

- With POST, refreshing the page involves forwarding again the information. It is necessary to alert the user.
- GET requests can be encoded directly into hyperlinks without a form.
  - <A href= "catalogo.php?id= 231222 "> See </A>

Activity 1. Forms and methods

- http://labit601.upct.es/asignaturas/ai/form.html
  - With the inspector examine the possible fields and make changes.
  - What are the names of the variables that are sent to the server?
- http://labit601.upct.es/asignaturas/ai/get.html
  - Capture the request with Wireshark and examine the packet. How is it coded the user information in the HTTP package?
- http://labit601.upct.es/asignaturas/ai/post.html
- http://labit601.upct.es/asignaturas/ai/archivo.html
  - Repeat the previous exercise for both URL.

Processing client information

In principle, the HTTP server will only attend requests: gets the requested resource and returns it to the client.

An external application should be responsible for processing client information and generating a result.

- How can an external application be run by the server?
- How the user information received by the server is passed to the application?
- Is there a standard mechanism?

Common Gateway Interface (CGI)

Standard procedure for communicating HTTP servers and external applications to dynamically generate documents.

- The URL of the request identifies an external executable program: the web server must be able to identify it with some criterion (extension, route, etc.).
- The server will run the external program and pass the user information.
- CGI specifies **how to pass parameters and run** external applications.
- Specifies the **Interface** between server and application
  - The external application can be programmed in any language, provided that it is able to accept input as specified by CGI, and create an appropriate response.
- It is very inefficient and has become obsolete

CGI operation

An external application and a HTTP server are two independent processes: problem of communication between process

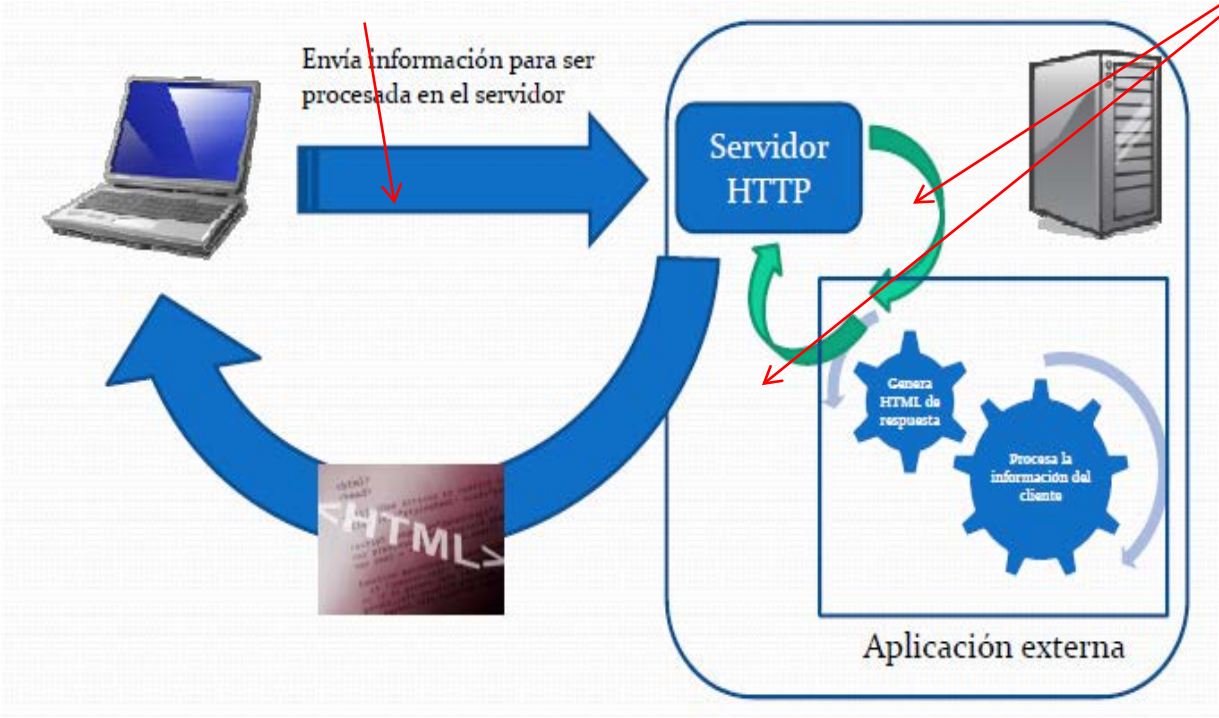
1. The server receives a request with a number of parameters (info from the user).
2. The server recognizes that the URL is not a document but an external application.
3. The external application is executed by the server in a special context:
   - It sets the value of a number of **environment variables**
   - For GET requests, the QUERY_STRING environment variable takes the value of the parameters as they appear in the URL.
   - For POST requests parameters are passed from the standard input (stdin).
   - Redirects the standard output (stdout) of the external application to himself (the server)
4. The information the server receives from the standard output (ie from the external application) is sent to the client:
   - Previously headers are added if necessary

Where is the interface that defines CGI?

CGI defines the interface between the HTTP server and the external application

Customer information arrives
- With GET: in the query string the URL
- With POST: encoded in the HTTP packet

CGI: how the HTTP server communicates with the external application



Information processing on the server

In practice, applications that process requests are integrated modularly in the HTTP server.

- Integrated processing languages used on the server: PHP, ASP, JSP and servlets, etc.
- They can access in an integrated manner to client information, server information, the request, etc. by special variables.
- The server identifies the type of request (usually in the URL) and launches the appropriate engine.

The engine or interpreter or executable is executed independently for each HTTP request.

- **No persistence of state or data.**
- It is necessary to implement it apart: with BBDD or cookies.

contents

State and data persistence

The HTTP protocol is a request-response protocol

- **There is not status information**: HTTP servers respond to each client without relating requests with previous or future ones.
- Each execution of a *script* or program as a result of a request is independent of others.

However, it is desirable to make certain applications set "sessions" that bring together multiple HTTP requests/responses in a larger context.

- We need to establish a state: to add additional information to indicate what is the state of the process / application / request.
- Who and how to store this information?
  - Databases: useful for permanently information but more complex applications, required for applications of certain size.
  - Cookies: simpler, but insufficient, and only for transient information, in general.

Sessions

Set of requests/responses related to each other in a given context.

Features of the sessions:

- Group a set of requests and replies
- The session is implicit in the exchange of status information
- They have a beginning and an end.
- A relatively short lifetime.
- The sessions can be ended by the server or the user agent (browser).
- **Typically are implemented with cookies**

Examples:

- "Shopping Cart": where the user chooses in different requests, different items to buy at the end of the session.
- User Authentication: necessary to prevent the user from having to provide credentials for each request.

cookies

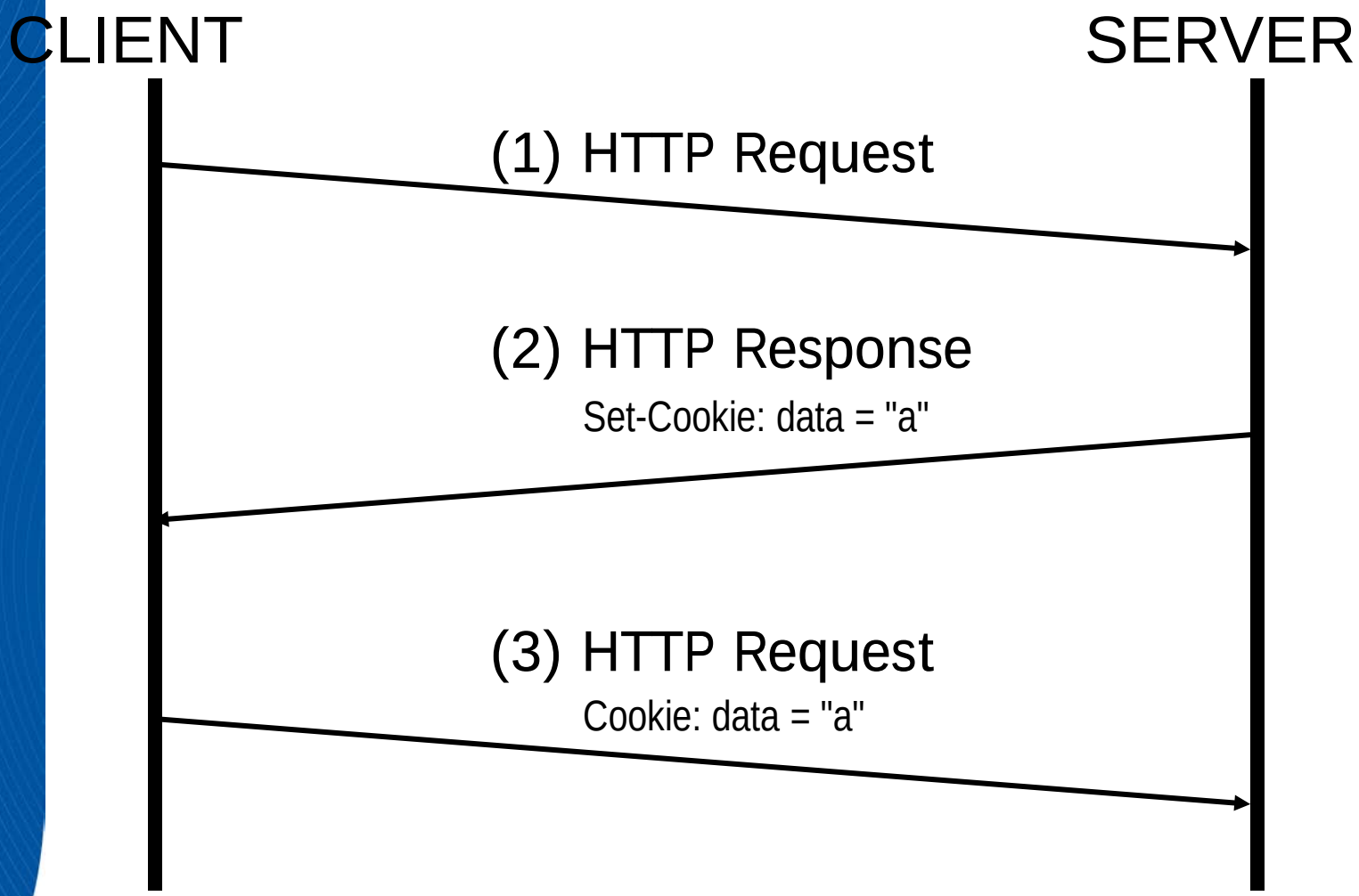
Cookies **is a technique that allows servers to send status information to the client and retrieve such information.**

- The server sends the client one or more **variables** with the corresponding value **that the client (browser) must store.**
- Each of these variables is a *cookie*.
- The customer returns them to the server in subsequent requests.
- Allow to implement sessions: they can be used as sesión or status ID

To do this, two new HTTP headers are used:

- In the replies: **Set-Cookie**.
- In the requests: **Cookie**.

Operation of cookies



Format cookies

Set-Cookie header contains a pair of name=*value*, which may contain modifiers separated by ";".

- The variable name may be anyone except those starting with $ which are reserved.
- The value can be any information that the server decides, its content is opaque to the client.

Attributes. Cookies can optionally include any of the following attributes. If not included the browser assigns the default value:

- *Comment:* the server reports the use of the cookie.
- *Domain:* It indicates the domain for which the cookie is valid.
- *Max-Age*: Validity of the cookie in seconds, afterwards is discarded.
- *Path:* Indicates the subset of URLs affected by the cookie.
- *Secure:* It indicates that the client should use a safe environment to contact the server (HTTP over SSL or similar). Additionally, one can set the attribute *HttpOnly*, Indicating that the cookie can only be modified by HTTP requests (no Javascript or other).
- *Version*: The version of the specification.

Managing cookies on the client

Interpretation of the Set-Cookie header by the user agent:

- Cookies are stored separately according to the source server, ie cookies for each server, identified by the domain name, are grouped.
- The client can discard it (does not store it)
  - Not accepted by the user or do not meet certain safety conditions.
- If any attribute is not specified, it is assumed:
  - `Domain` = Host to which the request has been made.
  - `Max-Age` = Dismiss the cookie at the end of the execution of the user agent.
  - `path` = The URL of the last request up to the directory of the document, ie, to the right-most "/".
  - `Secure` = Not required, ie, the *cookie* can be sent over an insecure channel.

Managing cookies on the client

Replacement and disposal of cookies

- If it receives a *cookie* with the name equal to a pre-existing one, and `Domain` and `path` match exactly in both, the new *cookie* replaces the previous one.
- If the new cookie has `Max-Age` equal to zero, the new and the previous one are discarded, the server uses this method to close the session.
- The number of storable cookies is finite, therefore the user agent must remove any cookie when there is no room for a new one.
- When the duration of a *cookie* is passed, it is discarded. The browser deletes it.

Cookies sent back to the server

It is done with the *Cookie* header

- Example: *Cookie: name = Pepe; visit = 1;*
- That is, a set of name = value pairs separated by ";"

The cookies selected to be returned to the server in a request are those that verify:

1. The canonical name of the server matches the stored *Domain*.
2. The attribute *path* matches exactly the **prefix** of the URL to be requested, until the last bar on the right.
3. The cookie has not expired (attribute *Max-Age*).

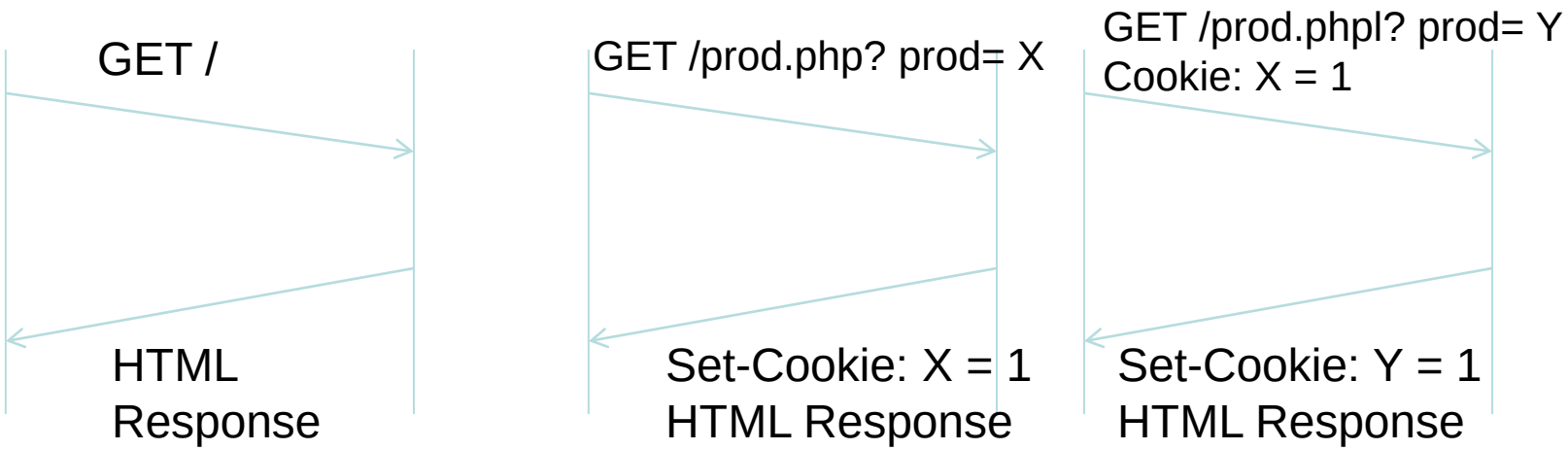
- If more than one cookie meets these criteria, they are sent in order, from the one with the more specific *path* to the least one.

Activity 2. Cookies

- http://ait.upct.es/asignaturas/ai/cookie.php
  - Check whether the visit counter works correctly, what value for the cookie *visit* sets the server? What value will have to show the user when you receive a cookie?
  - Check that the browser has stored cookies and their attributes. In Firefox Options> Privacy> Delete cookies.

- http://ait.upct.es/
- http://ait.upct.es/asignaturas/ai/ad/
  - For which of the above two requests the previous cookies have been sent to the server? Why?
  - If the server wants to reset the counter, how it is done? What would it send?

Example: “Shopping cart"



User requests catalogue

- Server replies with the catalogue in HTML

User selects product X

- Server adds to reply the cookie X=1
- Catalogue in HTML is shown again

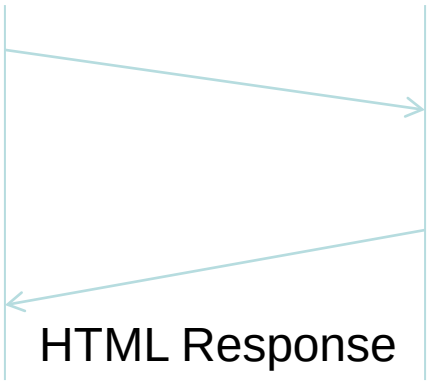
User selects product Y

- Server recives the cookie X=1 which was stored by the user
- Server send cookie Y=1
- Catalogue sshown again

Example: "Shopping cart "

GET /prod.php? prod= X
Cookie: X = 1; Y = 1

GET / comprar.php
Cookie: X = 2; Y = 1



User selects again product X

- Server receives cookies X=1and Y=1
- Server updates the number of products X with cookie X=2, that is, the value of cookie X is overwritten in the client

User decides to buy (proceed to checkout)

- Server receives all the previous cookies
- Looking at the values of cookies it knows that the user wants to buy 2 X products and 1 Y product
- Server returns the total price to be paid

contents

Databases (DB)

Definition: a set of data systematically stored for later use.

- How to store and process data more efficiently?
  - There are different models for implementing databases: relational, object, hierarchical

Database Manager : Application (program) that manages the structured storage and retrieval of data:

- Implements a database model.
- Standalone application (eg: MS Access).
- Client / Server model (eg: MySQL, Oracle, SQL Server, PostgreSQL, ...)

Relational Databases

DB model in which data is stored in tables that group together certain properties of data (fields).

- The columns identify the properties
- Each row is a separate record comprised of several fields.
- A table should group the characteristics of a system entity. Example: Users table, Products, Invoices ...

Each row of a table can be uniquely identified by the value of one or more columns: primary key

- Sometimes it is necessary to add artificial fields for a single (or effective) identifier.

Relationships between different tables are set by

- Inserting a primary key in another table
- Creating a table with primary keys of different tables
- Example: Tables Workers Tasks and WorkersTasks

Example

id	title	description	country	urlfoto
1	White House	During World War II (1939-1945), Cas ...	U.S	images / Casablanca.jpg
2	Blade Runner	At the beginning of the century, the powerful Tyrell Cor ...	U.S	images/Blade_Runner.jpg

Column: represents a data field

dni	nick	first name
32355567	carlitros	Carlos
63552121	pepelu	Pepe

primary key

Example: relational tables

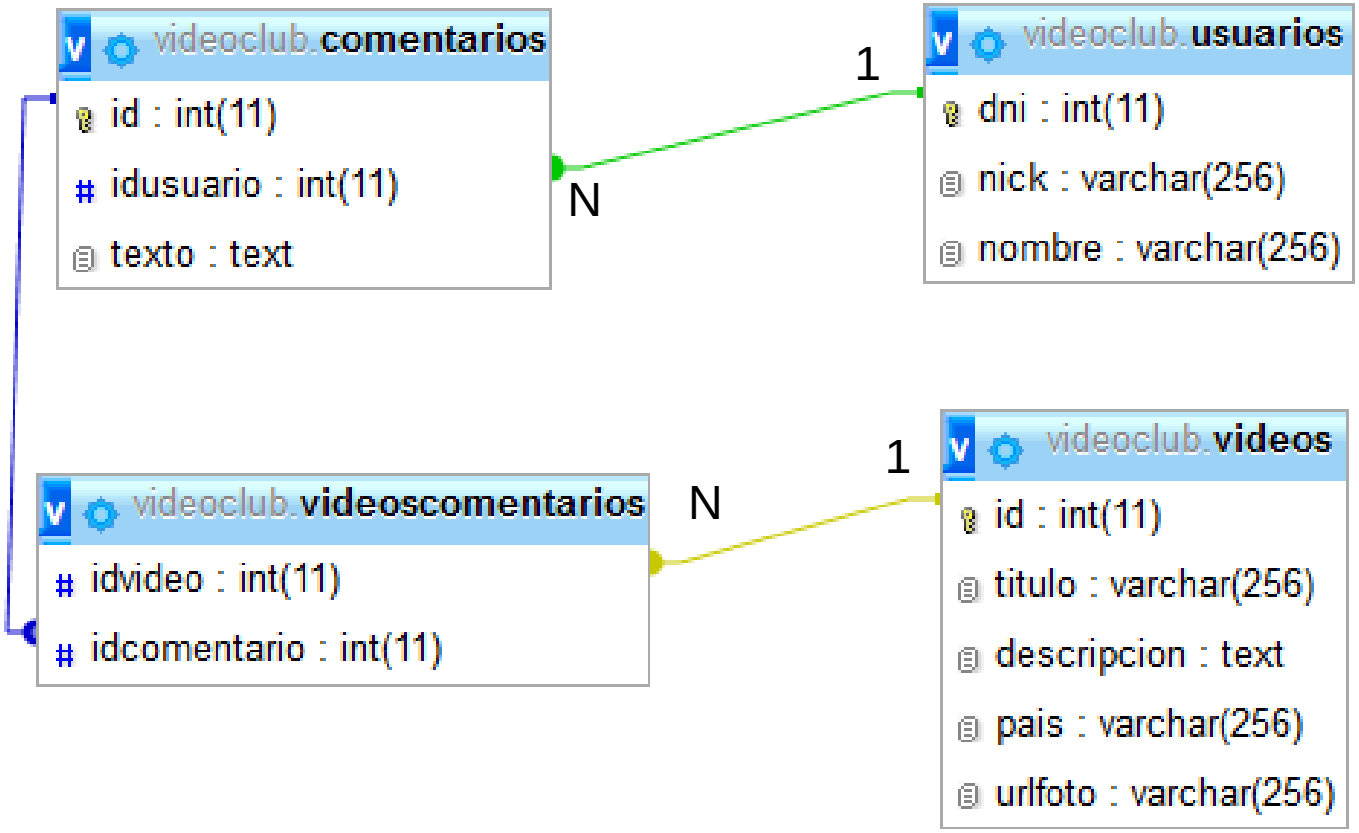
id	userid	text
1	32355567	buenisima
2	32355567	Passable
3	63552121	Masterpiece

idvideo	idcomentario
1	1
1	3
2	2

Column relates to the users table

relational table: relates two or more tables
Table related videos Table comments

Example: entity-relationship diagram



Relational databases: Normalization

Desirable features in a relational DB:

- The tables represent the data structure of the real problem.
- There should be no redundancy in the data.

Normalization: a design technique BBDD systematically eliminates redundancy in the data.

- First Normal Form (FN1): the values of each column are atomic (single value). Example: Name and surnames?
- FN2 and FN3: successive normalization rules. It is often desirable but sometimes introduces extra complexity.
- Define relationships types: 1-1, 1-n, nn

Database servers

Server program that manage databases, ie Database Manager.

- Clients connect to it and request data contained in the tables.

Services implemented:

- Indexing
- Concurrency control
- Transactions
- Security
- Additional functions

There is a standard language for querying a database server: SQL (*Structured Query Language*).

SQL

Structured Query Language: Standard language for querying a database server

- ANSI standard.
- There are different versions (extend the standard).
- Declares expressions, clauses, operators and predicates that allows queries on the data.
- Queries allow you to select "cells" of the tables, ie, obtain data as well as insert, delete or update data.
- It also allows "manage" the databases: create tables, users, assign permissions for viewing and modification, etc.

SQL: basic queries

Data Selection:

- SELECT fields FROM table WHERE condition
  - SELECT * FROM users WHERE id> 10 AND age> 25
  - SELECT title, author, publisher FROM books WHERE price <10 ORDER BY title

Data entry:

- INSERT INTO table (Field1, field2, ...) VALUES (val1, val2, ...)
  - INSERT INTO book (title, price, Editorial) VALUES ('test', 14, NULL);

Update data:

- UPDATE table SET WHERE field = value condition
  - Video UPDATE SET title= 'gremlims'WHERE id =' 10 ';

Delete data:

- DELETE FROM table WHERE condition
  - DELETE FROM books WHERE price > 15

SQL: select multiple tables

Selecting data from multiple tables: can be done in different ways

- JOIN, combines records from multiple tables in a query. Different types of JOIN:cross, inner, outer.
- The most common is the INNER JOIN: the result is a table with the fields of tables A and B for records that meet the condition
  - SELECT * FROM employee INNER JOIN department ON employee.DepartmentID = department.DepartmentID;
  - SELECT * FROM employee, department WHERE employee.DepartmentID = department.DepartmentID;
  - SELECT usuarios.nombre, comentarios.texto FROM usuarios, comentarios WHERE usuarios.id=comentarios.idusuario

SQL and programming languages

SQL is independent of the programming language.

- Queries are made by **SQL statements coded** as **strings** in the programming language.
- Many programming languages include libraries to work with relational databases.
  - For example, PHP provides functions to connect to SQL database servers and run queries
  - They also provide data structures and functions for working with the retrieved data: iterate, filter, etc.

SQL and PHP

```
<?php

    $ Link = mysqli_connect( 'localhost''mysql_user''mysql_password');
    if (! $ Link) {
        die ( 'Error Connection '. mysqli_error());
    }


    $db_selected = mysqli_select_db( "Books", $ link);
    if (! $db_selected) {
        die ( 'You can select BBDD'. mysqli_error());
    }

    $query = "SELECT * FROM books WHERE price> '$ price'";

    // Perform query
    $result = mysqli_query($query, $ Link);
    if (! $ result) {
        echo "No could make the query ($ Query) ". mysqli_error();
        exit;
    }
    // Loop through query
    while ($row = mysqli_fetch_assoc($result)) {
        echo $row["title"];
        echo $row["Author"];
        echo $row["editorial"];
    }
    mysqli_free_result($result);

    mysqli_close($ Link);
?>
```

The SQL query is a character string assigned to the variable $query

NoSQL Databases

In the last years other database models are increasingly used, especially for web applications and "big data".

They are primarily used to store unstructured information or weakly structured info with special features

- Object-based models: used since the 80s
- Document-based models: store heterogeneous documents (a set of properties and values) indexed by a key
- graph-based models: use graph structures

They have advantages and disadvantages: they tend to be more flexible or more efficient for particular types of applications but lack a standard query language and the practical and theoretical robustness of relational models

Document-based databases

- Store heterogeneous documents (a set of properties and values) indexed by a key
  - Each document can have a different number of properties
- Documents ("records" in relational databases) are organized into collections ("tables" in relational databases) and different views of the document (“queries" in relational databases) are provided
- Typically store data in specific formats: JSON, XML, binary
- Use programming languages as query languages: Java, javascript, C ++
- Usually they provide an API for **content-based queries**
- Apache CouchDB: http://couchdb.apache.org/
- MongoDB: http://www.mongodb.org/

Contents

Scripting languages on the client

Motivation: lack of interactivity with the user on the client

- The content (HTML) can not be dynamically modified as a result of user actions, unless a new request to the server is invoked and new content is received.
- HTTP operation forces to **execute a request and wait for the response for every user action.**
- Each user-generated "event" has to be processed by the server that returns an appropriate response

http://en.wikipedia.org/wiki/JavaScript

- List the possibilities of interaction we have with the previous page.

http://ait.upct.es/asignaturas/ai/get.html

- Again list how we can interact in this case.
- What happens in case the user makes a mistake in data entry?

Problems of the interaction model

With the usual operation of a browser and HTTP:

- Really few "events" available. Pretty much just click a link (or a submit button) = enter a new URL and request the document.
  - Very limited interaction with the user.
- It limits the development of rich user interfaces and degrades the "experience" of the user. The application is "less interactive."
  - Desktop applications have advantages in this regard: they have a very rich set of events to act upon.
- It does not allow the development of applications that change the content based on user activity on the window
- The server has to handle all processing performed, however simple (check that a field has been filled out correctly, for example), which can overload the server requests.

Solution: allow the execution of code instructions on the client

- Programmatic access and modification of the content displayed
- Programmatic access to the browser environment: capture events generated by user, appearance, behavior, functionality.
- Programmatic access to client resources: file system and peripherals. **Potentially very unsafe**

Javascript

Interpreted programming language whose interpreter is usually implemented by a web browser.

- Usually the interpreter is part of the browser: *Javascript engine*.
- The browser receives code (*scripts*) to be executed locally
- Allows scripts sent to the client as part of a document to:
  - **Interact with the user**: By executing instructions as a result of the occurrence of events generated by the user
  - **Modify the displayed content**.
  - **Communicate asynchronously** with the server.
  - **Control the browser functionality**.
  - **Access to the resources of the platform running the browser**: File System, Geolocation, Video, Audio, etc.
- **Sandardized as ECMAScript: ECMA-262 and ISO / IEC 16262.**

Operation

- The code to be executed by the client (*script*) is included with the document, using the HTML element **<script>:**
  - **Internal:** The element includes the code itself:
    - <Script> document.write("Hello World"); </ Script>
  - **External**: Includes a URL (absolute or relative) with the code to execute:
    - <script src="miscript.js“ type= "text/javascript" ></script>
- **Code is executed** in three different ways, according to the presence of the following attributes:
  - **Deferred execution**: is executed when the document has finished loading. Example <script src="demo_defer.js" defer></script>. **Only works with external scripts.**
  - **Asynchronous execution (new in HTML 5)**: immediately executes, **in parallel to the load** of the rest of the document. Example <script src="demo_defer.js" **async**></script>. **Only works with external scripts**
  - **Normal execution**: if the previous attributes are not present, the code is **inmmediately executed, before proceeding with the load and parsing** of the rest of the document.
- In many cases, the code is made of functions that will be executed when they are invoked, usually **when certain events occur.**

Funcionamiento (II)

Event-based execution: javascript code runs at an event occurrence

- A javascript function (previously declared) is associated with an event, and it is invoked when it occurs

```
<html>

<script>
      function say hello() {
            alert("Hello everyone!");
      }
</script>

<body onload= "say hello(); ">

</body>

</html>
```

Code of the function to be executed

EVENT: document fully loaded

Events

Event: action or event detected by the system that can be processed.

- There are a variety of defined events: standard events, as defined by W3C DOM events or non-standard events, defined by the browser engine.
  - Mouse events: click, dblclik, drag, mouseover, drop...
  - Keyboard events: keyup, keypressed, keydown..
  - Events related to the HTML document: load, unload, resize, scroll...
  - Form Events: submit, select, change, focus...
  - Events related peripherals, audio, video, DOM modifications, communications, etc.

Code assignment to events

Code assignment to events can be done in two ways:

- **HTML event attributes**: Are attributes of certain HTML elements that allow you to assign the execution of a function when they occur
  - <h1 onclick= "changetext(This) "> Click on the text! </ H1>
- **Allocation by code**: Using the DOM interface by javascript
  - <Script>
    document.getElementById("MyButton").onclick=validarFormulario;
    </ Script>
- In this case we must bear in mind that the item must already exist, or the call to *getElementById()* will return null. That is, the HTML document should have been processed *(parsed)* previously and that element exist.

Javascript languaje

- Languaje based on prototypes (objects that are cloned and dynamically extended), with dynamical types.
- Variable declaration and initialization: var y = 1; var c; c = 0;
- Dynamic types: no need to declare and a variable can accommodate different types on.
- Object-oriented: everything is an object.
  - var s = new String('Hello');
  - Access to methods and properties with "." var l =s.length;
  - An object can be declared dynamically also:

```
var myCar = new Object();
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
```

```
function  Car (make, model, year) {
 this.make = make;
this.model = model;
this.year= year; }
var c = new Car ( "Ford", "Fiesta", 1996);
```

Primitive types

Boolean

- *true* and *false*

Number

- 64-bit floating comma, similar to double in Java
- There are no integer
- *NaN* (Not a number) and *Infinity*

String

- Sequence of one or more Unicode characters
- Declared with ‘ or “

Special values

- *null* and *undefined*
- *typeof(null) = object;* *typeof(Undefined) = undefined*

Objects

An object is a collection of properties, where each property has a name and a value.

- var student = {name: "Pepe“, dni: 323222};
- Objects are mutable: the number of properties may change. You can add members at any time
  - alumno.nota= 6.5;
- They can have methods (functions) and *this* can be used to refer to an instance.
- They may include other objects as properties

The arrays and functions are also objects

- A property of a object can be a function (=method)
  - alumno.suma = function (a, b) {Return a + b};
- A function defines an object that can be added properties and even new methods:

```
function max (x, y) {if (x> y) return x; else return y;};
max.description = "Return the maximum of two arguments";
```

Creating Objects

- Using an instance of Object: the Object prototype is extended.
- Through what is called object initializer
  - It is equivalent to calling first new Object()
- Using a constructor function (declared previously)
  - The name of the constructor **always** must begin with capital letters

```
var myCar = new Object();
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
// Get an instance of the object
myCar
```

```
var myCar = {
 make "Ford“,
 model "Mustang",
 year = 1969
};
//We obtain an instance of object
```

```
function Car (make, model, year) {
this.make = make;
this.model = model;
this.year= year; }
//We have to instantiate he object:
var c = new Car ( "Ford", "Fiesta",
1996);
var d = New Car( "Fiat", "Bravo",
1990);
```

Inheritance

Javascript is a prototype-based language, which means that objects inherit directly from other objects, their *prototype,* no other classes

- All objects have a property *prototype* indicating the prototype object
- Objects inherit properties (and methods) of prototype
- All objects inherit directly from Object

The Object.create() method can be used to indicate the prototype of a particular object

```
var Animal = {
type: "invertebrates"
displayType : function() {console.log (this.type);
}
}

// Object created from prototype
var animal1 = Object.create(Animal);
animal1.displayType ();

// We extend the prototype
var fish = Object.create(Animal);
fish.type = "Fishes";
fish.displayType();
```

Functions

Declaration

- Usual
- Anonymous
  - Very useful for implementing *callbacks*
  - It is allowed to pass functions as arguments to other functions

```
function square(number) {
        return number * number;
 }
```

```
var square = function(number) {return
number * number};
var x = square(4)

document.getElementById("miboton").onclick
= function(){ alert('confirme envio');}
// assign the function
 as callback
```

```
function map(f,a) { var result = [];
 for (var i = 0; i != a.length; i++) {
 result[i] = f(a[i]); }
 return result; }

map(function(x) {return x * x * x}, [0, 1, 2, 5, 10]);
```

Functions

Parameters

- Primitive types by value. Everything else, by reference
- Any number of parameters
  - The object *arguments* is *as an* array with arguments
    - *nombrefun.length* number of arguments in the statement
    - *nombrefun.arguments.length* arguments **really** passed
  - It is not checked if the number of parameters passed matches the declaration

Scope

- Variables declared inside a function **are only accessible to the function.**
- If you used *var c* to declare a variable within a function, the variable will be local. However, if *var* not used, it will seek a global scope c variable and if it does not exist a variable c is created with global scope.

With the constructors of the object Function you can dynamically create a function from a string

- http://labit601.upct.es/asignaturas/ai/dynfun.html

When a function which is a property of an object (i.e., a method) is invoked *this* points to the object in question. When it is not a property, *this* refers to the global object

Control structures, loops, logical operators ...

```
for (var i = 0; i < 9; i++) {
        n += i;
        myfunc(n);
 }
```

Lists the properties of an object

```
for (var prop in obj) {
alert(objName + "." + prop + " = " + obj.prop + "\n";)
 }
```

```
var n = 0; var x = 0;
while (n < 3) {
        n++; x += n;
}
```

```
if (cipher_char == from_char) {
        result = result + to_char;
        x++;
} else {
        result = result + clear_char;
 }
```

Activity 3. Javascript

- http://labit601.upct.es/asignaturas/ai/js.html
  - Examine the source code of the page. What form of code execution has been used?
- http://labit601.upct.es/asignaturas/ai/jsevent.html
  - Examine the source code of the page. What events are used?
  - How these events are assigned?
  - Why the event onmouseover image is assigned when the event onload is executed?
  - What if the onmouseover event is directly assigned, as it appears, in the element <script>?
  - Use the *scratchpad* inspector from Firefox to make the figure resize when you click on it

DOM and javascript

The main goal of javascript is to dynamically modify a document displayed by the browser

The standard way of working with a structured document (HTML, XML, ..) is the DOM interface (Level 3)

Most browsers provide a javascript implementation of standard DOM Level 1, 2, 3 plus their own additional features

- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

With the javascript DOM API the contents of an HTML document can be changed dynamically (the changes are reflected immediately)

The document object

The *document* object and the interface *Document* are usually the gateway to the DOM and contains the most commonly used methods

- https://developer.mozilla.org/en-US/docs/Web/API/document

Most modern implementations also provide the interface *HTMLDocument*

Allows to access elements, in a global way, of the document: for example, all your stylesheets, all your images, hyperlinks, the URI

- *Document.links*, Document.location...

Allows to start browsing the document

- *Document.documentElement*
- *Document.children*

Browsing the document

The elements of a document implement interfaces like *Node*, *Element* and other refinements like *HTMLElement* which allow to navigate the tree

- children, childNodes, nextSibling...

Navigation is relative to the element on which a function is invoked

- *e.getElementsByTagName("P")* provides all P elements descending from the *e* element on which the call is made.
- *document.getElementsByTagName("P") returns all P elements of the document.*

In most cases a collection of items is returned

```
var children = e.childNodes;
for (var i = 0; i < children.length; i++) {
        console.log (children [i].nodeValue);
}
```

Modifying the document

- With the DOM functions in the interfaces Node, Element...
- Directly with *innerHTML*

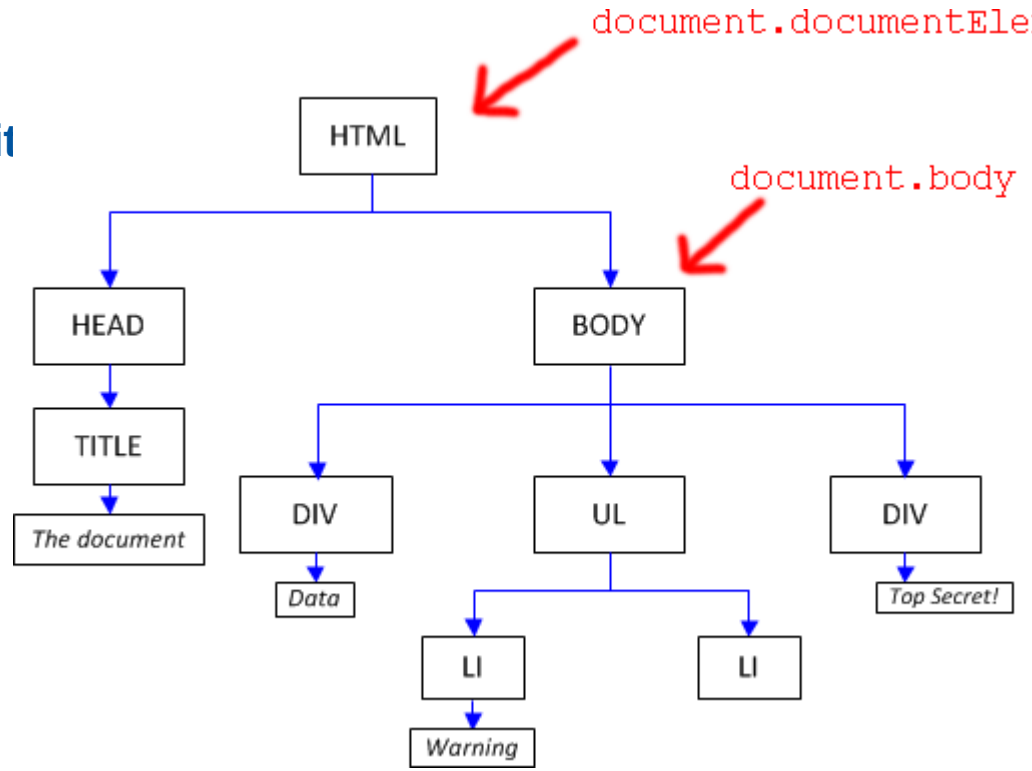
```
var p = document.createElement( "P");
p.setAttribute( 'class " big');
var e =document.getElementById( "intro");
e.appendChild(P);
```

```
var e =document.getElementById( "intro");
e.innerHTML = '<P class= "big"> New
paragraph</ P> ';
e.innerHTML + = '<Table>';
```

- *document.getElementById* and *document.getElementsByClassName* in the *document* object a,d *Element.getElementsByTagName* in elements and *document* are much used

Examples

```
<Html>
    <Head>
    <Title> The document </ tit
</ Head>
<Body>
<Div>
Data
</ Div>
<ul>
    <li> Warning </li>
    <li> </li>
</ul>
<Div>
Top Secret!
</ Div>
</ Body>
</ Html>
```



1. For the BODY element, indicated by arrows: parentNode, firstChild, lastChild, children, previousSibling
2. For the UL element show with arrows nextSibling, previousSibling
3. How do you get with Javascript all the DIV elements?

Example

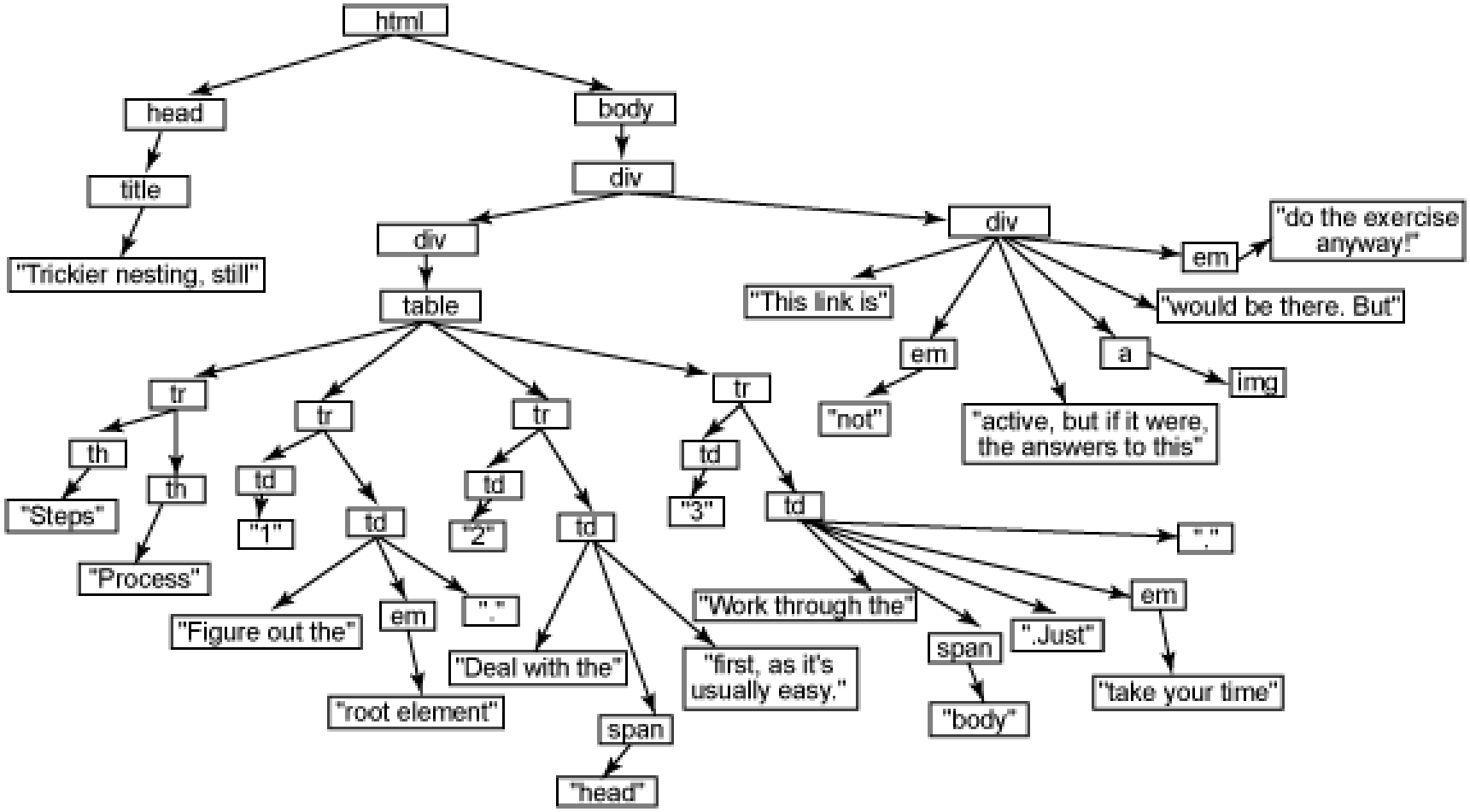
```
<Html>
<Head>
<Title> trickier nesting, still </ title>
</ Head>
<Body>
<Div id = "main-body">
<Div id = "contents">
<Table>
<tr> <th> Steps </th> <th> Process </th> </tr>
<tr> <Td> 1 </ td> <td> Figure out the <em> root
element </em>. </ Td> </tr>
<tr> <Td> 2 </ td> <td> Deal with the <span id = "code"> head </ span>
first,
Usually it's as easy. </ td> </tr>
<tr> <Td> 3 </ td> <td> Work through the <span id = "code"> body </ span>.
Just <em> Take your time </em>. </ Td> </tr>
</ Table>
</ Div>
<Div id = "closing">
This link is <em> Not </em> Active, but if it Were, the answers
to this <a href= "Answers.html"> <imgexercise.gif "/> </a>
would be there. but <em> Do the exercise anyway! </em>
</ Div>
</ Div>
</ Body>
</ Html>
```

- Draw the corresponding DOM tree for this HTML code
- Show this code in a browser

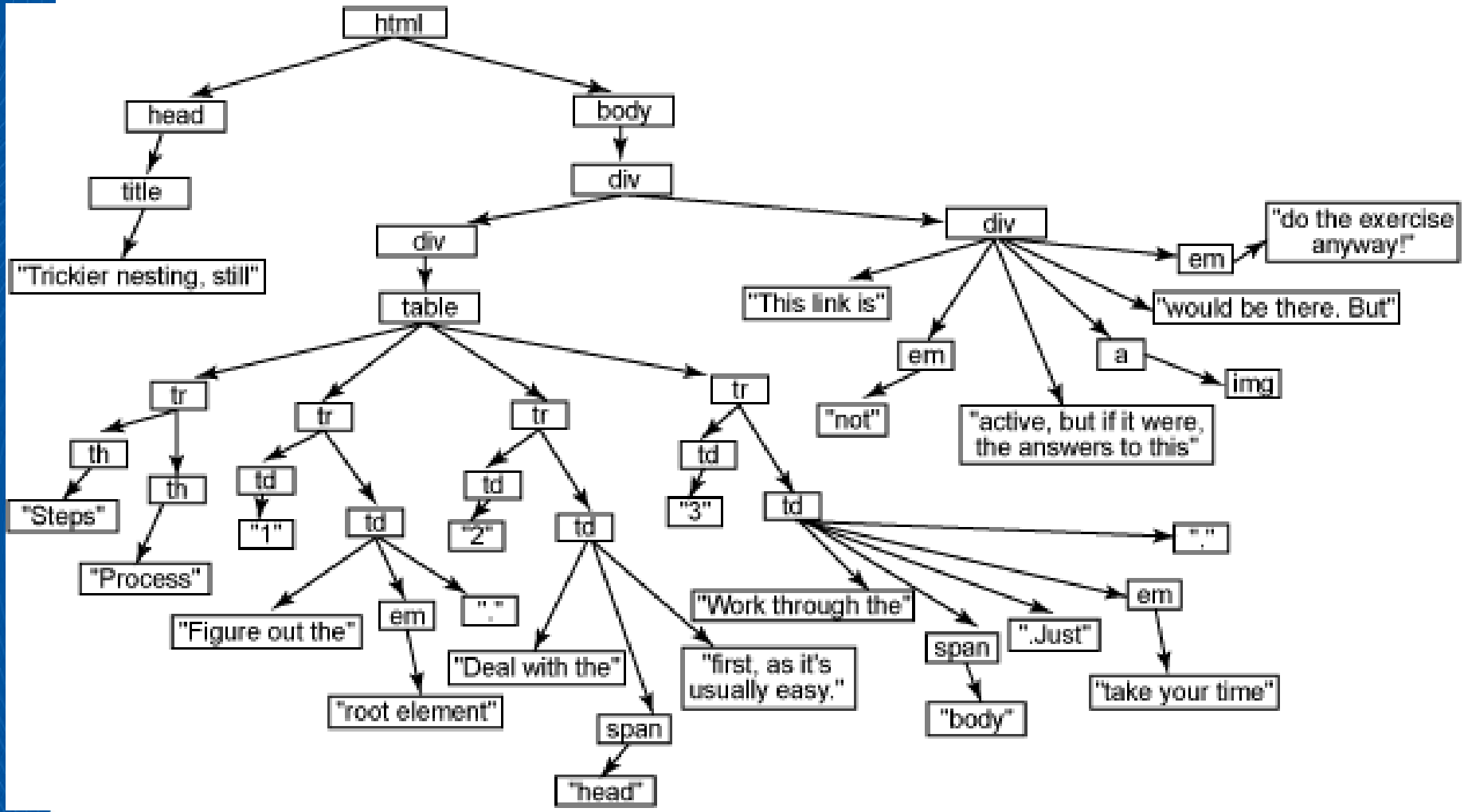
Example

1. Get all the columns of the document
2. Get only the columns corresponding to the third row

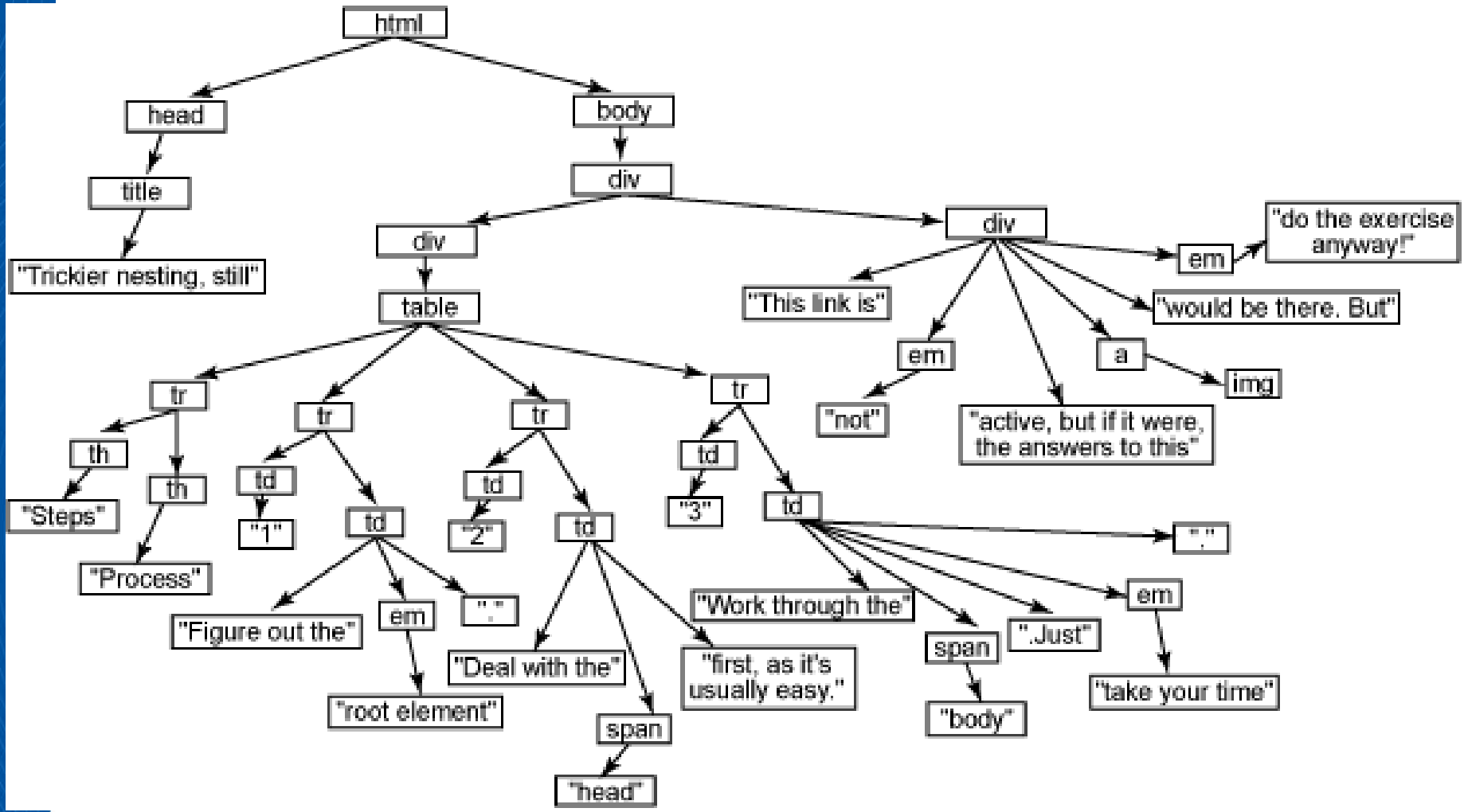
You can use scratchpad, along with the previous HTML document ...



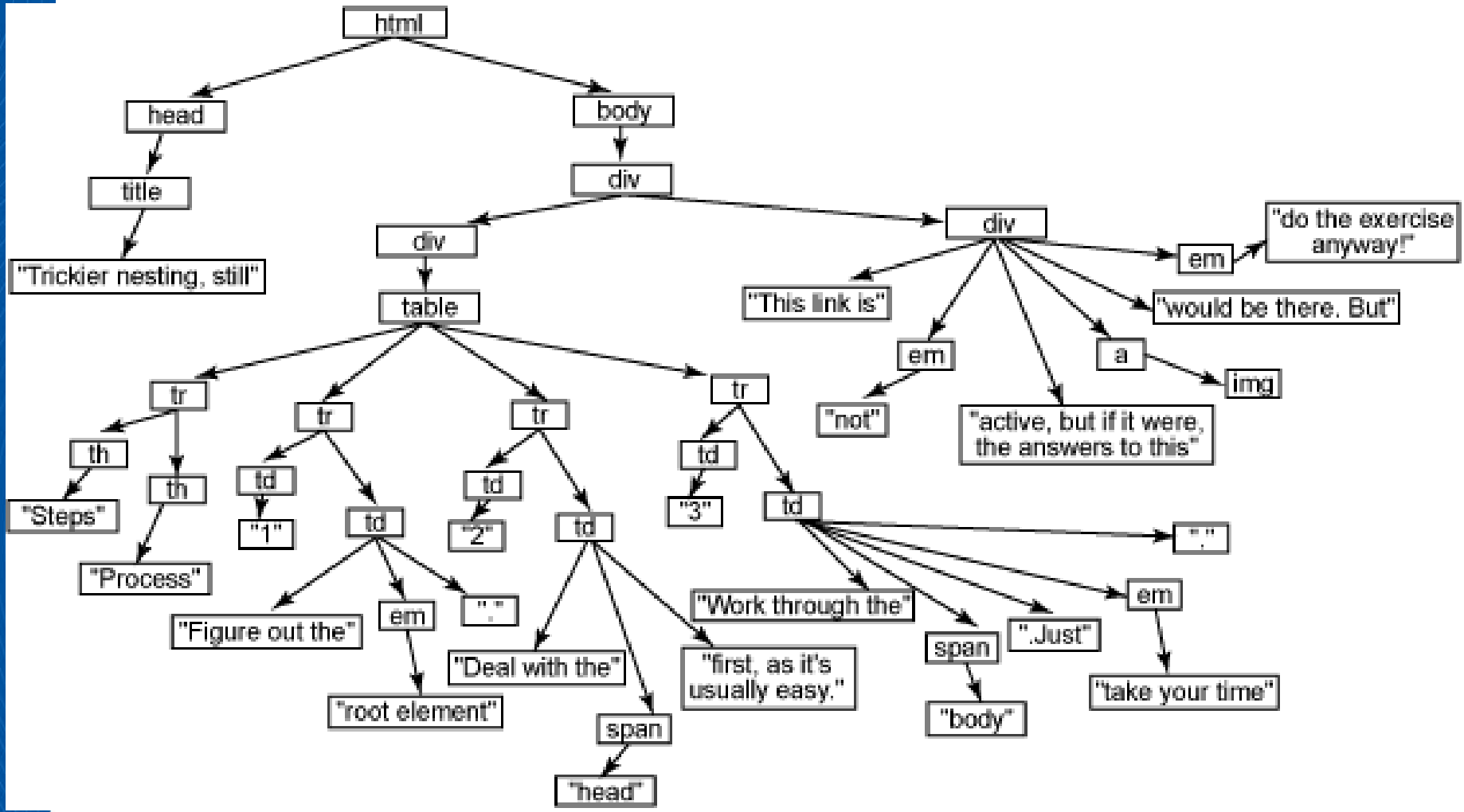
Example


html
head
body
title
div
"Trickier nesting, still"
div
div
table
"This link is"
em
"do the exercise anyway!"
"would be there. But"
em
a
img
"not"
"active, but if it were, the answers to this"
tr
tr
tr
tr
th
th
td
td
td
td
td
"Steps"
"Process"
"1"
"2"
"3"
"."
"Figure out the"
em
"."
"Deal with the"
"root element"
span
"head"
"first, as it's usually easy."
"Work through the"
span
".Just"
em
"body"
"take your time"

Example


html
head
body
title
div
"Trickier nesting, still"
div
div
table
"This link is"
em
"do the exercise anyway!"
"would be there. But"
em
a
img
"not"
"active, but if it were, the answers to this"
tr
tr
tr
tr
th
th
td
td
td
td
td
"Steps"
"Process"
"1"
"2"
"3"
"."
"Figure out the"
em
"."
"Deal with the"
"root element"
span
"head"
"first, as it's usually easy."
"Work through the"
span
".Just"
em
"body"
"take your time"

Example


html
head
body
title
div
"Trickier nesting, still"
div
div
table
"This link is"
em
"do the exercise anyway!"
"would be there. But"
em
a
img
"not"
"active, but if it were, the answers to this"
tr
tr
tr
tr
th
th
td
td
td
td
td
"Steps"
"Process"
"1"
"2"
"3"
"."
"Figure out the"
em
"."
"Deal with the"
"root element"
span
"head"
"first, as it's usually easy."
"Work through the"
span
".Just"
em
"body"
"take your time"

AJAX

Asynchronous Javascript and XML (AJAX): combined use of web technologies to make requests to an HTTP server without user intervention

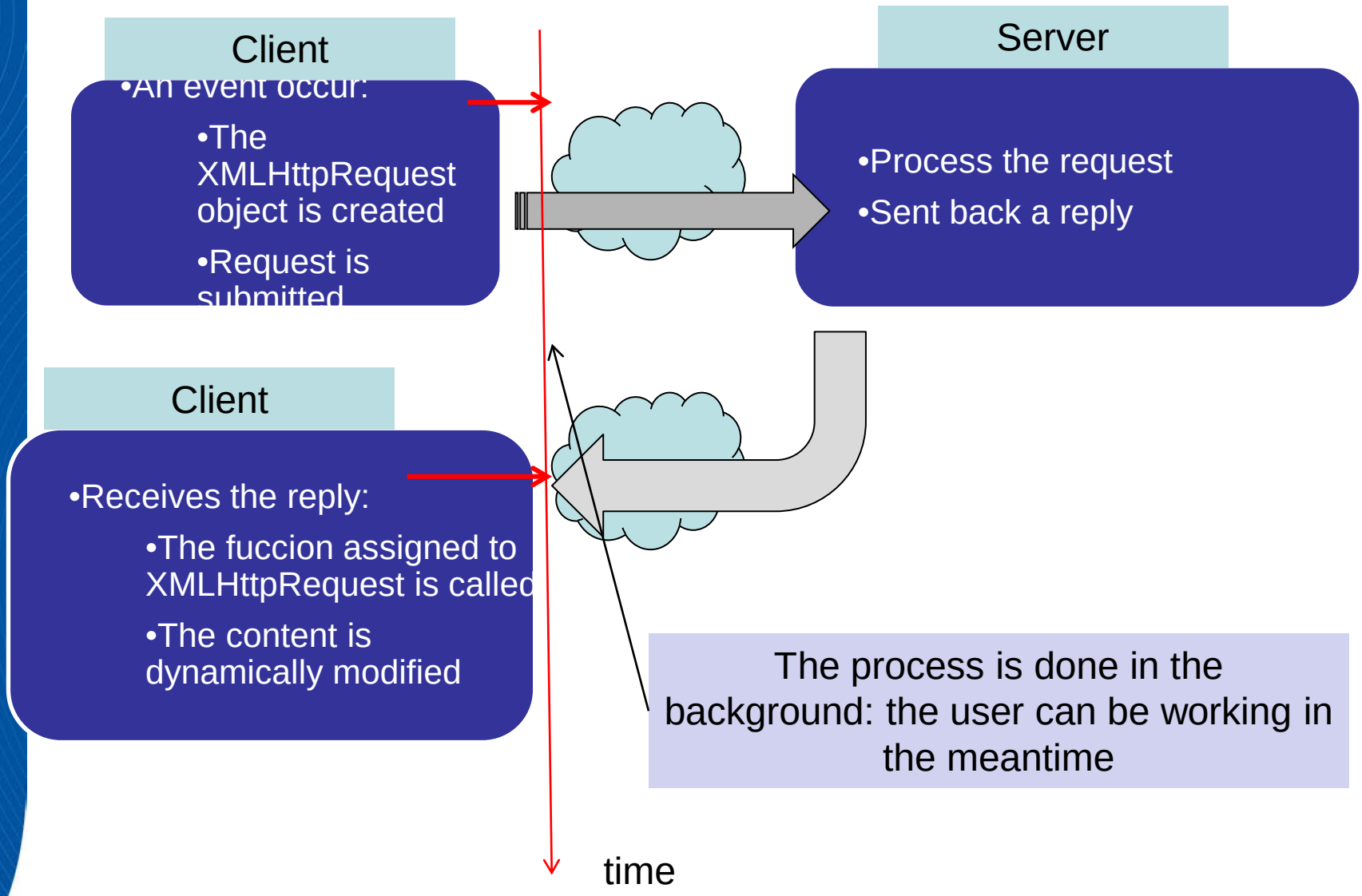
- It allows **asynchronous requests** (in the background or in parallel) to a web server using javascript code.
- When the **answer is available**, a preassigned function of javascript ***(callback function*)** is executed***.***
- The response (data sent by the server) can be used in conjunction with DOM and CSS to update the contents of the document dynamically.
- Originally response data were supposed to be encoded in XML format, but today it can be used with HTML, json and other formats.

AJAX operation

- Use the **javascript object XMLHttpRequest**.
  - The requested URL and type of request (GET, POST, ...) is indicated.
    - var xhr = new XMLHttpRequest();
    - xhr.open('get',' Send-ajax-data.php ');
  - The function that will process the response when available is assigned.
    - xhr.onreadystatechange = procesarRespuesta;
    - xhr.responseType = "text"; // Data Type Expected
  - The request is sent.
    - xhr.send();
- When the **answer** is **available** the **browser invokes** automatically the **function assigned to the request:**
  - The actual response status is checked locally
  - The result of the request (server response code) is checked
  - The data are processed

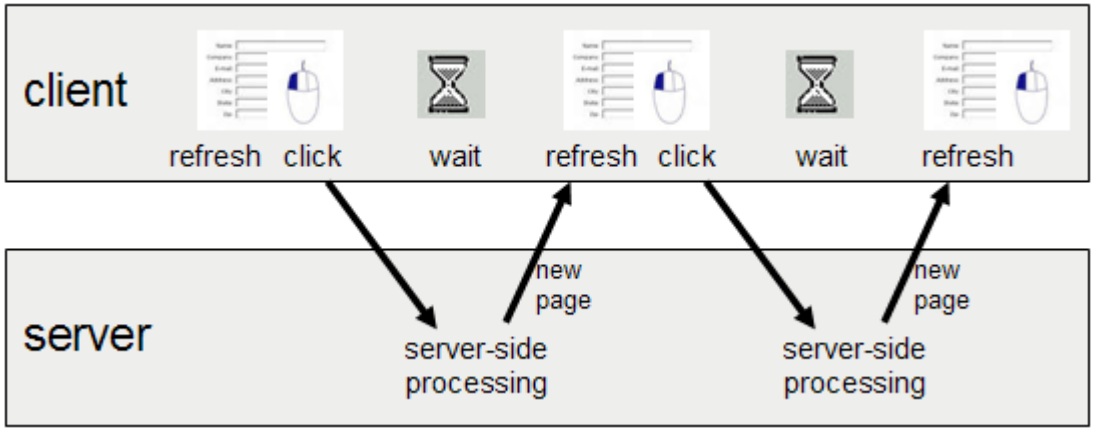
```
function procesarRespuesta() {
  if (xhr.readyState === 4) {
                if (xhr.status === 200) {
                  alert(xhr.responseText);
                }  else {
                    alert('problema.');
            }
        }
}
```

AJAX: Operation model

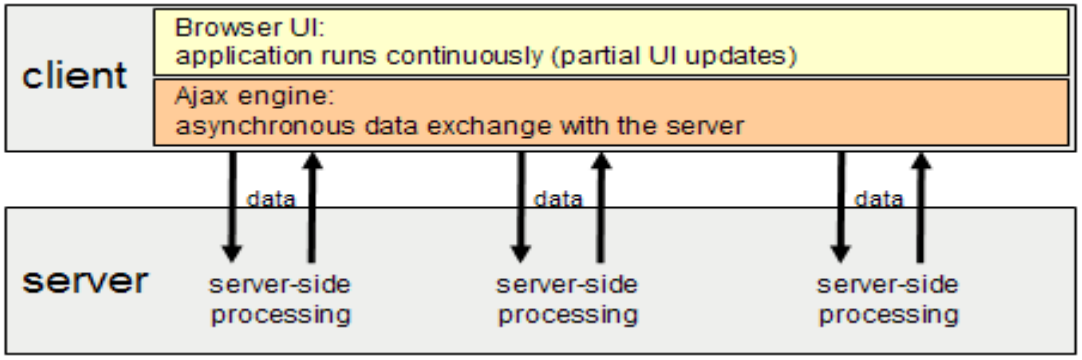


AJAX: Asynchronous model

traditional model



AJAX: Asynchronous



AJAX: final considerations

- AJAX is nothing more than a useful extensión of the event processing model of javascript: when a certain event occurs, the content of the document is updated by code, only that the new content is previously obtained from a server.
- Before HTML 5, AJAX calls were not recorded in your browser history: "back" button loses its functionality.
- Sometimes processing in the background creates drawbacks as the user is unaware of it: warn the user if possible.
- Alternatively: there is the **WebSockets** protocol (draft RFC 6455) that maintains a **bidirectional** connection for an indeterminate time with an HTTP server.

Activity 4. AJAX

http://labit601.upct.es/asignaturas/ai/ajax.html

- Examine both the source document and the javascript associated.
- Capture with wireshark or use the inspector (recommended) of your browser to see what happens when selecting a continent.
- Observe where the AJAX request is directed (you can find the source code in the file ajax.php.txt in the same directory) and notice that it does not have to match the *action* of the form.

- Look at the Exchange of packets with the inspector for the Example 2 of Activity 3 above.
  - Is AJAX used at some point? Was the new figure available on the client before passing the mouse over the prior one? Another request was sent?
  - In view of the observed, when is needed / recommended to use AJAX?

Quiz

- How is customer information sent to the server with HTML? What elements usually contains a FORM?
- What does the browser display if it dinfs an <input type= "Radio" yam= "Sex" value= "male">Male </ Input>?
- How is the name of the variables that the user has to fill in HTML forms specified?
- What are the differences between the GET and POST requests when sending data?
- What is a relational database?
- What is SQL?

issues

- indicate ways to add javascript code to an HTML document.
- Describe how and when javascript code execution occurs when the browser receives a document that incorporates it.
- Describe ways to assign events to a javascript function. What precautions should be taken?
- Explain how it Works the code evaluator http://labit601.upct.es/asignaturas/ai/dynfun.html
- How the *init()* function is assigned to the *load* event using HTML?
- Define AJAX.
- Briefly describe the steps necessary to obtain a text using AJAX and to display it in the browser.
- Can you add by javascript new elements to the HTML tree of a document already loaded by the browser? How does this work?

Bibliography and resources

- http://www.w3schools.com/sql/ Examples and tutorials simple SQL
- http://jsfiddle.net/ It allows testing code javascript directly on the browser.
- https://developer.mozilla.org/en-US/docs/AJAX Gecko AJAX documentation and tutorials
- http://jquery.com/ jQuery is a javascript library containing both user interface elements as well as varied functionality. Simplifies development of javascript
- http://prototypejs.org/ Another library that simplifies the development of javascript. More structured than the previous
- http://flightjs.github.io/ Library that provides an environment for web programming and javascript based on events.