

Módulo: CGI

Objetivos formativos:

Los objetivos formativos que aprenderemos a lo largo de esta unidad son los siguientes:

- Saber **qué es** un programa cgi y cuando se **ejecutan** éstos en los servidores web.
- Aprender cual es el **funcionamiento** de un programa cgi.
- Conocer cuales son las **tecnologías** que podemos usar para la implementación de cgi.

Pantalla: 1. ¿Qué es Cgi?

Introducción

Cuando el **World Wide Web** inició su funcionamiento como lo conocemos, empezando a tomar popularidad aproximadamente en 1993, solo se podía apreciar **texto, imágenes y enlaces**. La introducción de **Plug-ins** en los navegadores permitió mayor interactividad entre el usuario y el cliente, aunque estaba limitado por la velocidad y la necesidad de tener que bajar e instalar cada plugin que se necesitara, por lo que estos se desarrollaron mayormente en áreas de vídeo, audio y realidad virtual. El **CGI** (Por sus siglas en inglés "**Common Gateway Interface**") cambió la forma de manipular información en el web.

Esta tecnología tiene la ventaja de correr en el servidor cuando el usuario lo solicita por lo que es **dependiente del servidor** y no del ordenador del usuario.

Cuando un usuario rellena un formulario y envía los datos, esos datos se envían a un **servidor** donde se encuentra un programa **que recibe los datos de entrada y los procesa** para realizar con ellos cualquier tipo de operación tales como:

- ? Búsqueda de esos datos en una base de datos.
- ? Inserción, modificación o eliminación de esos datos en una base de datos.
- ? Envío de mensajes.
- ? etc

Una vez que esos datos han sido **procesados** el programa genera una salida indicando los resultados de la operación, que pueden ser:

- ? Resultados de la búsqueda en la base de datos.
- ? Mensaje de operación realizada con los datos...

Este programa que recoge los datos y los procesa y que se encuentra en el servidor **se llama CGI**

Con lo cual podemos definir **CGI (Common Gateway Interfaz)** como:

DEFINICIÓN

Un **interfaz que recoge los datos que un cliente** (el usuario que navega) **le envía, los procesa y genera una salida** (mensaje, acción...)

EJEMPLOS

Algunos ejemplos de CGI's que frecuentemente utilizamos son los siguientes:

- ? **Buscadores**. Introducimos la palabra o texto a buscar en la casilla del formulario. Pulsamos el botón "buscar". Ese dato llega al servidor y lo procesa, es decir, busca en su base de datos las entradas que contengan esa palabra o texto, ¿y qué salida genera?, pues la página web que nosotros vemos con los resultados de esa búsqueda.

- ? **Mensajes a móviles.** Seguro que más de uno ha utilizado las páginas web desde las que podemos enviar mensajes a móviles. Lo que hacemos es rellenar los datos y pulsar el botón, a continuación, esos datos son enviados y llegan a un servidor. Éste los procesa y envía un mensaje al móvil indicado, y la salida es la página web que nosotros vemos donde nos dice: "Su mensaje ha sido enviado".

Pantalla 2: ¿Cómo funciona un programa Cgi? (i)

Veamos como funciona una cgi.

Los pasos que ejecuta una cgi son los siguientes:

1. Un usuario que se encuentra navegando por Internet, rellena un formulario de una página web.
2. Pulsa el botón Enviar y envía los datos al servidor donde se encuentra la cgi
3. La cgi recoge los datos, los procesa y genera una salida que el usuario ve en su ordenador.

A continuación vamos a ver el funcionamiento de una CGI ilustrado con un ejemplo.

Para ello vamos a seguir los siguientes pasos:

Fot 1.- En primer lugar acudimos al buscador "Google". (<http://www.google.es>)



Aquí nos encontramos un formulario de búsqueda para que el usuario, en este caso nosotros, podamos "insertar", es decir, interactuar con la web, lo que deseamos buscar.

Fot 2. Este formulario, tiene los siguientes campos que podremos rellenar:

- ? **Campo de texto.** El campo que se encuentra a continuación del texto "Find this". Este campo será donde introduzcamos el texto o palabras que deseamos buscar.
- ? **Búsqueda:** Un conjunto de "radio button", o botones de opción, para decidir donde realizar la búsqueda.

Como vemos en este caso, no en todos los campos de un formulario podemos escribir. En algunos escribimos, pero en otros seleccionamos la opción u opciones que nos ofrezcan. En cualquier caso, son campos de formulario donde el usuario debe interactuar para introducir datos o seleccionar alguno de los propuestos.

Fot 3. Por último, y aunque no lo hemos dicho todavía, en todo formulario existe el "botón de acción". Este botón será el que pulsemos para comenzar a procesar los datos. Por cuestiones puramente estéticas a veces este botón será una imagen.

En nuestro caso el botón de acción es:

Búsqueda en Google

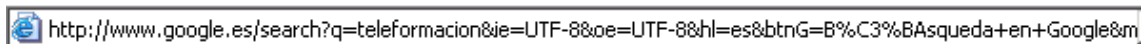
Fot 4. Ahora que ya estamos ante una página web con un formulario, llega el momento de "interactuar" con éste. Introducimos en el campo de texto lo que deseamos buscar:



Hemos introducido el texto "Teleformación", hemos escogido páginas en español para la búsqueda, y hemos seleccionado la opción "La Web" para buscar páginas web.

Fot 5. Finalmente pulsamos el "botón de acción", que este caso es el botón **Búsqueda en Google**.

Si nos fijamos en la barra de direcciones una vez que pulsemos el botón de acción, veremos lo siguiente:

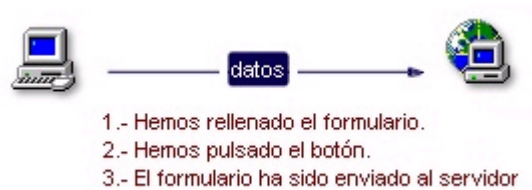


Si nos fijamos en la barra de herramientas, podemos leer y deducir lo siguiente:

- ? Los datos del formulario han sido enviados al servidor **http://www.google.es**
- ? El programa CGI que ha recogido los datos se llama **"search"**.
- ? Al programa "search" le han llegado como parámetros los datos que anteriormente introdujimos en el formulario. Esos datos son los que se encuentran a continuación del signo ?

Pantalla 3: ¿Cómo funciona un programa Cgi? (ii)

En el ejemplo anterior nos hemos encontrado con la siguiente situación :



Así, los datos viajan desde el ordenador en el que los hemos introducido a través del formulario hasta el servidor.

En el servidor, se encuentra un programa **CGI** que recoge esos datos y realiza con ellos la operación para la que dicho programa CGI ha sido creado.

Veamos a continuación cómo ha funcionado en nuestro ejemplo anterior el programa CGI llamado "search".



- ? El programa CGI "search", que se encuentra en el servidor "http://www.google.es" **recibe** los datos del formulario que nosotros rellenamos
- ? El programa CGI "search" está diseñado para **buscar** en la base de datos de direcciones web que tiene, las que contengan la palabra "Teleformación" y estén en idioma español, que son los datos que ha recibido de nuestro formulario
- ? La base de datos **devuelve** la lista de páginas web que contengan la palabra "Teleformación" y estén en español
- ? El programa CGI "search" recibe esa lista y con ella genera una página web con la apariencia determinada. Esta página web es la que nos **envía** a nuestro ordenador y es la página web que vemos de resultados de la búsqueda.

Pantalla 4. Tecnologías de implementación

Ahora que ya sabemos qué es CGI y conocemos su filosofía de funcionamiento, vamos a enumerar algunas de las más importantes tecnologías de implementación para diseñar y desarrollar programas CGI 's.

<i>Tecnología</i>	<i>Descripción</i>	<i>Ventajas</i>	<i>Inconvenientes</i>
<i>SQL - C</i>	Desarrollos de CGI en lenguaje C, incrustando en cada momento la sentencia SQL necesaria	Rápidez, robustez y la amplia difusión del lenguaje C.	Labor tediosa a la hora de generar la página con los resultados.
<i>Perl</i>	Lenguaje interpretado para generación de CGI	Multiplataforma	Ha quedado obsoleto pues otras tecnologías han evolucionado con mayor difusión y rapidez
<i>PHP</i>	Lenguaje de scripts de servidor	Robusto y orientado a Unix	
<i>ASP (Active Server Pages)</i>	Lenguaje de scripts de servidor	Facilidad y versatilidad	

Pantalla 5. Resumen

Ideas claves

A lo largo de esta unidad habrás aprendido que:

Cgi en sí, es un método para la transmisión de **información** hacia un compilador instalado en el **servidor** y su función principal es la de añadir una mayor interacción a los documentos web que por medio del HTML se presentan de forma estática.

El **CGI** es utilizado comúnmente para **contadores, bases de datos, motores de búsqueda, formularios, generadores de email automático, foros de discusión, chats, comercio electrónico, juegos en línea** y otros.

El programa cgi se encargará de **procesar** y dar **respuesta** a un envío de datos realizado por un usuario tras rellenar éstos desde un formulario web.

Conocer cuales son las **tecnologías** que podemos usar para la implementación de cgi en servidores web.

Módulo: ASP

Ud1. Asp y VbScript

Pantalla Objetivos formativos:

Los **objetivos formativos** que aprenderemos a lo largo de esta unidad son los siguientes:

- Conocer aspectos fundamentales empleados en Asp como su origen y finalidad, así como los **componentes necesarios** para poder desarrollar aplicaciones o la manera de poder visualizar unos resultados.
- Aprender los conceptos básicos empleados en la programación Asp, centrándonos especialmente en el lenguaje **VbScript**, lenguaje del que se vale Asp para la implementación de cualquier aplicación web.
- Conocer distintas funciones incluidas dentro de VbScript agrupadas por categorías, para terminar finalmente en la inclusión de archivos con código.

Pantalla: 1. ¿Qué es Asp?

Definición:

ASP es la alternativa de **Microsoft** a los CGI's y Java Servlets. Al igual que éstas, se trata de una **tecnología** que se aplica **en el** lado del **servidor** para la **generación dinámica de contenidos Web** y que permite la interacción con otros sistemas, entre ellos el email y los ficheros.

Desarrollo:

La programación en ASP se basa en la creación de **archivos** con la extensión **'.asp'**.

Para ello, basta cualquier editor de texto en ASCII, incluido el 'Block de Notas' de Windows.

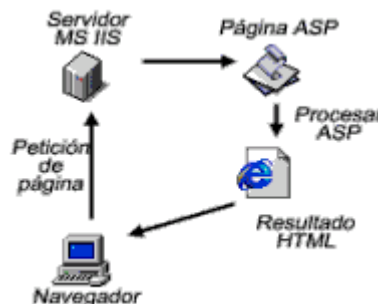
Un programa Asp se **compone** de los siguientes elementos:

- ? Texto.
- ? Etiquetas HTML.
- ? Secuencias de comandos en VBScript o JScript.

Las **secuencias de comandos** no son más que un **conjunto de instrucciones** escritas en un **mismo lenguaje de programación**, en este caso, **VBScript** o JScript.

Mas info1:

Cuando se ejecuta una secuencia de **comandos**, dichas instrucciones **se envían a un intérprete** (que debe estar habilitado **en un servidor de Web**), que los interpreta y posteriormente los ejecuta. El **resultado de la ejecución** se devuelve al cliente que solicita la página en formato **HTML**.



Esto implica, por un lado, la **ocultación del código fuente al cliente**, y por otro, la **no** existencia de **restricciones** especiales **en el** lado del cliente (**navegador**) para la visualización de las páginas.

En la parte del **servidor**, sin embargo, se emplea un archivo (**asp.dll**) para la interpretación del código, archivo que se distribuye con los **servidores de Microsoft (Internet Information Server o Personal Web Server)**.

Asimilación

ASP es una tecnología ejecutada en el lado del **servidor de Web** que permite generar **contenido dinámico** Web a través de páginas **'.asp'**, elaboradas con **etiquetas HTML** convencionales **y secuencias de comandos** en un lenguaje denominado **VBScript**.

Pantalla 2. El servidor web

La **ejecución** de una página **'.asp'** debe hacerse **siempre a través** de un **servidor** de Web. En caso contrario, no se obtienen resultados, al no realizarse ni la interpretación ni la ejecución de los comandos.

Para ello, las páginas deben estar alojadas en algún directorio, generalmente virtual, que forme parte del servidor.

DEFINICIÓN

Un **directorio virtual** es un directorio que **lógicamente forma parte del servidor** de Web pero que **físicamente puede estar** ubicado **en cualquier lugar**, ya sea junto con el resto de carpetas que se crean en la instalación, en otra parte del disco duro o incluso en otra máquina.

Algunas Aclaraciones

Cada **directorio virtual** del servidor tiene asociado **un alias**.

Definición o desplegable de alias

Un **alias** es un **nombre con el** que los clientes (**navegadores**) deben **acceder a los directorios virtuales** y con el que se oculta, también, la posición física real en el servidor del directorio y de los archivos contenidos.

Cuentan además, con una serie de **permisos** (lectura, escritura, ejecución, secuencias de comandos o exploración de directorios), que **determinarán lo que se pueda hacer o no** con los ficheros que almacenen.

Más info2:

El **permiso mínimo** requerido para un directorio virtual que vaya a contener **páginas '.asp'** es el de **ejecución de secuencias de comandos**.

Para la visualización de **páginas '.html'**, **imágenes** u otro elemento de lectura, se requiere que se habilite dentro del servidor el permiso de **lectura** en ese directorio.

De manera inicial, en la estructura del servidor que se instala por defecto, aparece habilitado un directorio virtual, situado **en C:\InetPub** denominado **Scripts** preparado **para páginas '.asp'**.

El directorio **Scripts** suele tener **deshabilitado el permiso de lectura**, por lo que las **páginas '.html'** y las **imágenes** deberán **situarse** en un directorio preparado para ello, que en la estructura que se instala por defecto es **c:\inetpub\wwwroot**, cuyo alias es **/**.

Asimilación/Ahora ya sabes

Las **páginas '.asp'** deben colocarse en algún **directorio que forme parte del servidor de Web** y que tenga habilitado dentro del mismo el **permiso de ejecución de secuencias de comandos**.

Las **páginas '.html'** y las imágenes deben ir colocadas en un **directorio que forme parte del servidor de Web** y que disponga del **permiso de lectura**.

Pantalla 3. El servidor de Web

Veamos las posibilidades de carga de páginas a través del servidor de web.

Visualizar una página '.asp'

Para **visualizar** una página **'.asp'**, debe hacerse a través del servidor de Web. Para ello, en el navegador, **debe cargarse la página, llamando al servidor Web** que la tiene alojada. Esto se consigue escribiendo en la barra de direcciones una URL con el siguiente formato:

- ? <http://localhost/aliasdirectoriovirtual/pag.asp>
- ? <http://nombremaquina/aliasdirectoriovirtual/pag.asp>
- ? <http://dirIPdelServer/aliasdirectoriovirtual/pag.asp>

Si utilizamos la estructura del servidor de Web por defecto, las páginas **'.asp'**, que deberán estar alojadas en el directorio **C:\inetpub\scripts** cuyo **alias** es **/scripts** se cargarán con:

- ? <http://localhost/scripts/pag.asp>
- ? <http://nombremaquina/scripts/pag.asp>
- ? <http://dirIPdelServer/scripts/pag.asp>

Visualizar una página '.html'

Si lo que queremos es **visualizar** una página **'.html'** en la estructura que se instala por defecto con el servidor Web, deberemos tener colocadas **las páginas en el directorio C:\inetpub\wwwroot**. A continuación no hay más que escribir en la barra de direcciones del navegador cualquiera de las siguientes alternativas, siempre teniendo en cuenta que el **alias** de este directorio es: **/**.

- ? <http://localhost/pag.html>
- ? <http://nombremaquina/pag.html>
- ? <http://dirIPdelServer/pag.html>

Visualizar una imagen

Para cargar una **imagen**, ya sea en una **página '.asp' o en una '.html'** en la estructura del servidor que se instala por defecto, la **imagen debe estar en directorio C:\inetpub\wwwroot**. Para hacer referencia a la imagen con el **alias** del directorio, que en este caso es **/**.

```
<IMG src="/miimagen.gif">
```

O bien

```
<IMG src="/imagenes/miimagen.gif">
```

si la imagen estuviera dentro de un **directorio imágenes en wwwroot** (C:\inetpub\wwwroot\imagenes)

Pantalla 4. Principios básicos de Asp y VbScript

Reglas sintácticas básicas:

- ? Dentro de una página asp, las secuencias de comandos deben ir entre los **delimitadores especiales** `<% %>`.
- ? Sólo debe haber una sentencia por línea ya que el **separador de sentencias** es el **retorno de carro**.
- ? VBScript **no distingue** entre **mayúsculas** y **minúsculas**.
- ? La **interpretación** y ejecución se realiza **a medida que se va analizando** el archivo.
- ? Para realizar una salida de datos, y permitir que los mismos sean enviados al navegador, es decir, para **colocar** algún tipo de **resultado en el código HTML** que se va a devolver al cliente, los datos deben colocarse en una **expresión de salida**, que se identifica por estar entre los siguientes símbolos `<%= %>`.
- ? La **integración** entre el código en **HTML** y en **VBScript** es **total**. Puede haber sentencias de VBScript dentro de etiquetas HTML y a su vez, las etiquetas HTML, pueden formar parte de sentencias complejas en VBScript (sin olvidar nunca que cada tipo debe estar entre sus delimitadores específicos).
- ? Los **comentarios** se introducen precediendo cada línea por una **comilla simple** (`'`).

Todas las **variables** se definen bajo un mismo **tipo** de datos denominado **VARIANT**. Dicho tipo, permite la utilización directa de las variables sin importar la naturaleza que tengan. Lo cual implica que **pueden emplearse directamente** sin necesidad de declararse ni de inicializarse.

Si no se indica lo contrario, una **variable**, dentro de un **contexto numérico**, se **inicializa** al valor **cero** y en un **contexto de cadenas**, adquiere el valor de **cadena vacía**.

Como alternativa, es **posible declarar las variables** mediante la palabra clave **Dim** e incluso **forzar la declaración de todas** y cada una de ellas mediante la sentencia **Option Explicit**.

Todos los **literales de cadenas de texto** van entre comillas (`"`), mientras que los de **fechas y horas** entre almohadillas (`#`). Los literales numéricos se escriben sin ningún tipo de modificación.

EJEMPLOS

1.

```
<HTML>
<HEAD>
<TITLE>Ejemplo de página Asp</TITLE>
</HEAD>

<BODY>
<%
    x=2
    cadena="hola"
    ' Al introducir una expresión de salida a continuación hay
    ' hay que cerrar el delimitador debido a que las expresiones
    ' de salida deben de llevar sus propios símbolos de apertura y cierre

    ' la siguiente línea mostraría el valor de la variable x
%>

<FONT color="red"><%= x %></FONT>
```

```
</BODY>  
</HTML>
```

2.

```
<HTML>  
<HEAD>  
<TITLE>Ejemplo de página Asp</TITLE>  
</HEAD>
```

```
<BODY>  
<% Option explicit  
    Dim w,x, y, z  
        w=2  
        x="Probando"  
        y=#11/11/2003#  
        z=#10:00:00 pm#  
%>  
</BODY>  
</HTML>
```

Pantalla 5. **Sentencias de control de flujo. Condicionales**

VBScript presenta instrucciones para manipular la secuencia en que se deben ejecutar las instrucciones:

? **Sentencia IF**

La estructura de la **sentencia condicional**, implementada mediante la instrucción **IF** presenta el siguiente formato:

```
<% If condicion then
instrucciones
[ElseIf condicion then
[InstruccionesElseIf]]
[Else
[InstruccionesElse]]
End If %>
```

? **Sentencia Select Case**

La **selección** entre una serie de opciones la realiza la sentencia **Select Case**:

```
<% Select case expresion
[case caso1
[instrucciones caso 1]]
[case caso2
[instrucciones caso2]]
.....
[case caso n
[instrucciones caso n]]
[case else
[instrucciones para el resto]]
End Select %>
```

EJEMPLO

1. Sentencia If

```
<HTML>
<HEAD>
<TITLE>Ejemplo de sentencia if</TITLE>
</HEAD>
<BODY>

<%if x<3 then %>
  <%= "x tiene un valor inferior a 3" %>
<% elseif (x=3) and (x<=5) then %>
  <%= " x tiene un valor entre 3 y 5" %>
<% else %>
  <%= " x tiene un valor superior a 5" %>
<% end if%>

</BODY>
</HTML>
```


2. Sentencia Select Case

```
<HTML>
<HEAD>
<TITLE>Ejemplo de sentencia Select Case</TITLE>
</HEAD>
<BODY>

<%Select case provincia
  case ="Málaga"
    id="Ma"
  case="Sevilla"
    id="Se"
  case="Cádiz"
    id="Ca"
  case else
    id="Otros"
End Select
%>
Procedencia: <%=id%>

</BODY>
</HTML>
```

Pantalla 6. Sentencias de Control de flujo. Bucles

Continuando con las sentencias que modifican el flujo secuencial de un programa, nos encontramos con los bucles, que nos permiten ejecutar un bloque de sentencias reiteradamente, mientras que se cumpla una determinada condición.

Podemos construir un bucle en Asp, mediante las siguientes instrucciones:

While

El **bucle while** presenta el siguiente formato:

```
<%do while condicion
Instrucciones while
[Exit Do]
Instrucciones while
loop %>
```

Do-Until

La estructura de la **iteración do-until** se presenta a continuación:

```
<% do
Instrucciones while
[Exit Do]
Instrucciones while
loop until condicion %>
```

For

Finalmente, el formato de un **bucle for**:

```
<% for contador=inicio To final [Step salto]
Instrucciones For
[Exit For]
Instrucciones For
next %>
```

EJEMPLO

1. While

```
<HTML>
<HEAD>
<TITLE>Ejemplo de bucle while</TITLE>
</HEAD>
<BODY>

<% x=1
do while x<=10
```

```
        x=x*2
    loop
%>
x: <%=x%>

</BODY>
</HTML>
```

2. Do-Until

```
<HTML>
<HEAD>
<TITLE>Ejemplo bucle do-until</TITLE>
</HEAD>
<BODY>

<% x=1
do
    x=x*2
until x<=10
%>
x: <%=x%>

</BODY>
</HTML>
```

3. For

```
<HTML>
<HEAD>
<TITLE>Ejemplo bucle for</TITLE>
</HEAD>
<BODY>

<% x=1
for i=1 to 5
    x=x*2
next
%>
x: <%=x%>

</BODY>
</HTML>
```

Pantalla 7. Procedimientos y Funciones

ASP permite también **reutilizar código** mediante el empleo de **Procedimientos y Funciones**.

Procedimientos

Los **procedimientos**, a diferencia de las funciones, se caracterizan porque cuando se les invoca **nunca devuelven ningún tipo de valor**. Tan sólo realizan una serie de acciones, sin más.

```
Sub miprocedimiento ([lista-argumentos])
sentencias
[Exit sub]
....
End sub
```

Lista-argumentos está compuesta por una serie de argumentos (si fueran necesarios) separados por comas y con la siguiente estructura:

```
[ByVal|ByRef] nombreargumento[( )]
```

ByVal, By Ref indican "Paso por Valor" y "Paso por Referencia", respectivamente. Por defecto, se toma ByRef

Una vez definido el código del procedimiento, para que las instrucciones que contienen se lleven a cabo, basta con llamarlo.

La **llamada a un procedimiento** se puede realizar de dos maneras:

```
Call miprocedimiento(lista-de-valores-para-los-argumentos)
```

O bien,

```
miprocedimiento lista-de-valores-para-los-argumentos
```

Funciones

Por otra parte, se encuentran también las **funciones**, que a diferencia de los procedimientos, **siempre devuelven un resultado** tras su llamada. La **devolución de resultados**, se realiza **asignando** a una **variable** que lleve el **mismo nombre que la función**, el valor de retorno.

```
Function mifuncion ([lista-argumentos])
sentencias
[Exit function]
...
mifuncion= ....
End function
```

Lista-argumentos está compuesta por una serie de argumentos (si fueran necesarios) separados por comas y con la siguiente estructura:

[ByVal|ByRef] nombreargumento[()]

ByVal, By Ref indican "Paso por Valor" y "Paso por Referencia", respectivamente. Por defecto, ByRef

Ejemplos

Procedimientos

```
<HTML>
<HEAD>
<TITLE>Ejemplo procedimiento</TITLE>
</HEAD>
<BODY>
' Definición del procedimiento

<% Sub mostramensaje(m) %>
    <%=m %>
<% End Sub %>

' Llamada al procedimiento
<%
    saludo="Hola"
    call mostrarmensaje(saludo)
%>

</BODY>
</HTML>
```

Funciones

```
<HTML>
<HEAD>
<TITLE>Ejemplo Funciones</TITLE>
</HEAD>
<BODY>

<% Function suma(x,y)
    suma=x+y
    End Function
' Llamada a la función y recogida
' del resultado que devuelve
    resultado=suma(2,10)
%>

</BODY>
</HTML>
```

Pantalla 8. Manejo de Cadenas

VBScript proporciona funciones para el **manejo** específico de **cadenas**.

Especialmente mencionable es el índice con el que se numera cada carácter que compone la cadena. A diferencia de otros muchos lenguajes, en VBScript, **el primer carácter** de la cadena **ocupa la posición 1**.

Además, entre los **métodos** de uso **más frecuentes** se encuentran:

Len(cadena)	Obtiene la longitud de una cadena
Instr(cad1,cadabuscar,inicio)	Localiza una cad a buscar dentro de cad1
Mid(cadena,inicio,longitud)	Extrae una subcadena
LCase(cadena)	Convierte la cadena a minúsculas
UCase(cadena)	Convierte la cadena a mayúsculas
Trim(cadena)	Elimina los espacios y tabuladores que puedan existir por delante y detrás de la cadena
cadena1 & cadena2	Concatena cadena1 y cadena2

EJEMPLO

```
<HTML>
<HEAD>
<TITLE>Ejemplo Cadenas</TITLE>
</HEAD>
<BODY>

<% cadena1="Hola que tal"
   cadena2=" que"

   if "que"=Trim(cadena2) then %>
       <%= "son iguales" %>
   <% end if
       ' Devuelve son iguales
       %>
       <br>
       <%=Ucase(cadena1) %>
       <% ' muestra HOLA QUE TAL
       %>
       <br>
       <%=Ucase(cadena1) %>
       <% ' muestra hola que tal
       %>
       <br>
       <%=Replace(cadena1, "tal", "haces") %>
       <%=cad %>
       <% ' muestra Hola que haces %>
       <br>

       <%if Instr(cadena1, "que")<>0 then %>
           <%= "encontrado"
       <%else%>
           <%= "no encontrado"%>
       <%end if
```

"Diseño de Páginas Web" - HTML Avanzado

```
'Devuelve encontrado %>  
</BODY>  
</HTML>
```

Pantalla 9. Inclusión de archivos.

ASP permite **incluir archivos** dentro de las páginas, con el fin de facilitar la **reutilización de código**.

Estos ficheros, suelen llevar, por convención, la extensión **'.inc'** y pueden contener **cualquier elemento** que pueda aparecer dentro **de una página ASP**, es decir, etiquetas en HTML, texto o sentencias en VBScript, cada cual con sus correspondientes delimitadores.

El archivo debe estar situado en un **directorio dentro del servidor** que cuente con **permisos de lectura**.

El contenido de los archivos **se inserta en el punto en que aparece la sentencia de inclusión**.

Esta tarea, puede realizarse de dos formas:

```
<!-- #include VIRTUAL ="path/archivo.inc" -->
```

o bien,

```
<!-- #include FILE ="path/archivo.inc" -->
```

Estas sentencias son el **único caso de código ASP** que **nunca** debe ir **entre los símbolos <% %>**

La distinción entre ambas viene dada por la forma de direccionar el archivo. Si para localizarlo se emplea un **direccionamiento virtual** (mediante el alias del directorio virtual donde se encuentra), se utiliza la palabra clave **VIRTUAL**

Si por el contrario se emplea un **direccionamiento relativo**, debe usarse **FILE**.

Un **archivo puede incluir**, a su vez **otros**, siempre y cuando la cadena de inclusiones **no genere un ciclo**. Llegado este caso, ASP emite un mensaje de error informando de la situación.

Finalmente, mencionar que estas sentencias (**#includes**), de existir, **son lo primero que se ejecuta dentro de una página ".asp"**.

RESUMEN

Una vez que has finalizado esta unidad deberías de haber aprendido que:

La programación en ASP se basa en la creación de **archivos** con la extensión **'.asp'**. Y que para la **ejecución** de una página **'.asp'** debe hacerse **siempre a través** de un **servidor** de Web.

1. Para **visualizar cualquier página dentro de un servidor** hay que **conectarse** primero al servidor de Web **mediante http://localhost, el nombre de la máquina o la dirección Ip de la máquina y** a continuación indicar el **alias del directorio virtual** al que se quiere acceder.

2. Las **sentencias** en VBScript deben ir entre los **delimitadores <% %>**.

3. Para poner el **resultado de alguna sentencia en el código HTML** que se envía tras la ejecución de una página ASP, éste debe colocarse en una **expresión de salida, <%= %>**.

4. **No es obligatoria la declaración de variables.**

5. Las constantes de **cadenas** van entre comillas (**"**), las de **fecha y hora** entre almohadillas (**#**) y las **númericas** no llevan **ningún símbolo** especial.

6. Asp dispone de **sentencias de control del flujo** de ejecución de programas, así como la posibilidad de creación de **procedimientos y funciones**. También posee diversas funciones para el **manejo de cadenas**.

7. **ASP** permite **incluir archivos con código** dentro de sus páginas. Estas inclusiones **son lo primero que se procesa** en una página ASP y **nunca** deben ir **entre** los símbolos **<% %>** de VBScript.

Módulo: ASP

Ud2. Objetos Integrados y Manejo de Ficheros

Pantalla Objetivos formativos:

Los **objetivos formativos** que aprenderemos a lo largo de esta unidad son los siguientes:

- Conocer todos y cada uno de los **objetos** que se encuentran disponibles en Asp.
- Aprender que éstos objetos añaden gran número de funcionalidades a ASP, debido a que incorporan un buen número de **métodos**, todos ellos bien clasificados, que facilitan en gran medida ciertas labores de programación dentro de las páginas ASP y de lo que es el entorno del servidor.
- Saber como manipular **archivos** de texto desde una página Asp. Tareas como la **creación, acceso, lectura o escritura** de ficheros conoceremos a lo largo de esta unidad.

Pantalla 1. Objetos Integrados

ASP cuenta con la presencia de una serie de objetos, que facilitan ciertas tareas de programación dentro del servidor de Web.

En todo lenguaje de programación orientado a objetos, para poder emplear uno cualquiera de ellos, primeramente hay que obtener lo que se ha dado en llamar instancia.

Un proceso de **instanciación** consiste básicamente en **dos pasos**:

- ? LLamar a algún **método que cree el objeto**.
- ? **Asignar** el resultado **a una variable**. (Dicha variable recibe el nombre de **instancia**).

En términos generales quedaría algo similar a:

variable = crearobjeto("tipo-del-objeto")

A diferencia de los objetos convencionales, **ASP** viene dotado también con **cinco objetos especiales**, denominados **Integrados**, cuya característica principal es que **no requieren ningún proceso de instanciación** para poder emplearse dentro de una página y usar todo lo que traen. **Directamente se pueden usar dentro del código** usando su nombre y la función o método deseados.

Estos **objetos integrados** son **cinco**, y agrupan métodos y propiedades de muy distinta naturaleza:

Request: Contiene métodos y propiedades que permiten **recuperar** algún tipo de **información que proviene del usuario** (los datos de un formulario, por ejemplo).

Response: Contiene métodos y propiedades que **envían algún tipo de información al usuario**.

Server: Contiene métodos y propiedades que **manipulan** ciertos aspectos de lo que es el **entorno del servidor**.

Session: Manipula las **sesiones de usuario**.

Application: Manipula una **aplicación**.

Más info:

Una **sesión de usuario** no es más que el **tiempo que transcurre** desde que el **usuario entra** en el servidor y accede a una de las páginas del directorio virtual **hasta que se va**.

En **ASP**, se denomina **aplicación** al conjunto de **páginas ASP** que se encuentran **dentro de un directorio virtual**.

Pantalla 2 Objeto Request

El primero de los objetos integrados en el que se va a profundizar va a ser el **Objeto Request**.

Definición

El **objeto Request** permite **recuperar** cualquier tipo de **información** incluida en una **petición HTTP**.

Algunas Aclaraciones

Estos datos, pueden ser de **origen muy diverso**:

- ? Un conjunto de **parámetros enviados** con el método **POST**.
- ? Un conjunto de **parámetros de consulta** adjuntos al método **GET**.
- ? Un conjunto de **datos estándar** incluido el juego de **variables del servidor**.

Asociada a cada tipo de información, ASP proporciona lo que ha dado en llamar **Colección**, a través de la cual se **permitirá el acceso a dichos datos**.

Así pues las **colecciones** más importantes de este **objeto Request** son:

Form: Recupera datos enviados mediante el **método POST**. (Datos de un formulario enviado con el método POST)

QueryString: Recupera datos enviados mediante el **método GET**. (Datos de un formulario enviado con el método GET o adosados a una dirección de salto en un hipervínculo).

ServerVariables: Obtiene información que se envía desde el cliente y que se encuentra **asociada a ciertas variables** dentro del **servidor** de Web.

La **sintaxis general** de acceso a la información tiene el siguiente formato:

Request.Colección(nombrevariable)

- ? **Request.form.**

Para **recuperar la información** proveniente **de un formulario** con **método POST**, basta acceder al **name del elemento del formulario** cuyo valor se quiere consultar. Esto es, si en el formulario existe un **cuadro de texto** con el atributo **name="email"**, para acceder al valor rellenado en este campo, sería suficiente con poner **Request.Form("email")**

En el formulario, previamente se ha debido especificar en el **atributo 'action'** que los datos, tras el envío, deben mandarse a la **página ASP que los va a procesar**.

```
<HTML>
<HEAD>
<TITLE>Ejemplo de página Asp. Página del formulario</TITLE>
</HEAD>
<BODY>
```

```
<!-- al pulsar el botón de envío los datos se mandan a procesar.asp
que será la página en que se traten -->
```

```
<FORM action="/scripts/procesar.asp" method="POST">
Nombre: <input type="text" name="nombre"><br>
email:<input type="text" name="email"> <br>
Sexo: H:<input type="radio" name="sexo" value="Hombre"> &nbsp;
      M:<input type="radio" name="sexo" value="Mujer"><br>
Empleado:<input type="checkbox" name="Desempleado" value="Si"><br>
Aficiones: <select name="aficion">
              <option value="Viajar"> Viajar</option>
              <option value="Música"> Música</option>
              <option value="Deportes"> Deportes</option>
              <option value="Libros"> Libros</option>
            </select><br>
      <input type="submit" value="Enviar">
</FORM>
</BODY>
</HTML>
```

La página "procesar.asp" que recibirá automáticamente los datos cuando se pulse el botón submit de envío quedará:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de página Asp. Página que recibe los datos</TITLE>
</HEAD>
<BODY>

<!-- dentro del request.form debemos pasdar entre paréntesis y comillas
      el name del elemento del formulario que queremos obtener -->
Nombre recibido: <%= Request.form("nombre") %> <br>
email:<%= Request.form("email") %> <br>
Sexo: <%= Request.form("sexo") %> <br>
empleado:<%=Request.form("Empleado") %> <br>
Desempleado: <%=Request.form("Desempleado") %><br>
Aficion: <%=Request.form("Aficion") %><br>

</BODY>
</HTML>
```

? Request.QueryString.

Para el caso del método de envío GET, las modificaciones serían mínimas:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de página Asp. Página que recibe los datos (procesar.asp)</TITLE>
</HEAD>
<BODY>

<!-- en el atributo action del formulario hay que colocar las páginas Asp
      que recibirán los datos cuando pulsemos el botón de envío-->

<FORM action="/scripts/procesar.asp" method="GET">
Nombre: <input type="text" name="nombre"><br>
<input type="submit" value="Enviar">
```

```
</FORM>
</BODY>
</HTML>
```

La página ASP (procesar.asp), que recibe los datos quedaría:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de página Asp. Página que recibe los datos (procesar.asp)</TITLE>
</HEAD>
<BODY>

<!-- Para recoger los datos de un formulario enviado con Get, se utiliza la
       colección querystring y entre paréntesis y comillas el name del elemento del formulario
       que se quiere recoger -->

nombre recibido: <%= Request.QueryString("nombre") %> <br>

</FORM>
</BODY>
</HTML>
```

? **Request.ServerVariables.**

Las **ServerVariables** pueden suministrar **información de tipo diverso**. Entre las más destacadas están:

REMOTE_ADDR	La dirección IP del host remoto que hace la petición.
SCRIPT_NAME	La ruta virtual donde se ejecuta el script.
SERVER_NAME	Devuelve el Nombre de Host del servidor.
SERVER_PORT	Número de puerto que usa el servidor.
REQUEST_METHOD	El método utilizado para mandar los datos del formulario HTML, estos pueden ser GET, HEAD, POST.
REMOTE_HOST	Nombre del host remoto que hace petición. Si el servidor no tiene esta información, pondrá la de REMOTE_ADDR y dejará la anterior vacía.
QUERY_STRING	Parámetros de la petición Get (Cadena que sigue al signo interrogante (?) en la petición HTTP).

Para **consultar** el valor de una **ServerVariable**, bastaría poner: **<% dato=Request.ServerVariables("NOMBRE de la VARIABLE") %>**

Pantalla 3. Objeto Response

Definición.

El **objeto Response** se emplea para **enviar distintos tipos de respuesta** de vuelta al cliente, es decir, **la página HTML** que se devuelve como **resultado**.

Algunas Aclaraciones:

Las propiedades y métodos más usados son las siguientes:

Response.Write(cadena): Escribe **cadenas** de caracteres sobre **la salida HTML** que se envía de vuelta. **Equivale a** la sentencia `<%= cadena %>`. La ventaja es que no requiere cerrar y abrir los delimitadores de VBScript antes y después de la sentencia de salida.

Response.Buffer: Determina si se empleará un **buffer a la hora de enviar el resultado al cliente**. Los valores: true ó false.

Response.redirect("pag.asp"): Envía una **redirección** (a la página indicada como argumento) **al cliente**. **Requiere** que lo primero que haya en la página donde se encuentre esta sentencia sea el comando **Response.Buffer = true**

Response.Clear: **Borra** el contenido del **Buffer** de salida.

EJEMPLO:

A continuación se muestra un ejemplo de código empleando **Response.write** y mostrando la diferencia con una expresión de salida convencional de tipo `<%= %>`

```
<HTML>
<HEAD>
<TITLE>Ejemplo de response.write</TITLE>
</HEAD>
<BODY>

<% if Request.form("nombre")="Daniel" then
    Response.write("Hola Daniel")
end if%>

<% ' A continuación la misma sentencia con <%= %>
' Observar como hay que cerrar y abrir los delimitadores <% %>
' antes y después del <%= %>
if Request.form("nombre")="Daniel" then
%>
<%= "Hola Daniel" %>
<% end if %>

</BODY>
</HTML>
```

Pantalla 4: Objeto SERVER

Definición

El **objeto Server** va a representar al servidor de Web. Proporciona una serie de métodos y propiedades relacionados con la funcionalidad básica del servidor tales como la creación de otros objetos o el manejo de directorios virtuales..

Algunas Aclaraciones:

Las propiedades y métodos más usados son las siguientes:

CreateObject: Permite crear instancias de objetos en el servidor.

HTMLEncode: Codifica un cadena de texto a codificación HTML. Es útil si se quiere escribir literalmente texto que contenga código HTML, sin que se interprete por el navegador.

MapPath: Traduce de ruta virtual a ruta física.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Ejemplo con Objeto Server</TITLE>
</HEAD>
<BODY>

<% ' Dado un archivo pag1.asp en un directorio virtual en el servidor con ruta
' "/Scripts/ejemplos" es posible calcular su dirección física mediante MapPath
%>

<%=Server.MapPath("/scripts/ejemplos/pag1.asp") %>
<% ' la función anterior devolvería: c:\inetpub\scripts\ejemplos\pag1.asp
%>

<br>

<%= Server.HTMLEncode("<font color=red> Rojo</font>")
<% ' La sentencia anterior sacaría literalmente la frase: <font color=red>Rojo</font>
' y no solo la palabra Rojo en color rojo como haría cualquier
' navegador al interpretar ese código HTML
%>

<% 'Creación de una instancia de un objeto para el manejo de conexiones a
' una base de datos

set bd= Server.CreateObject("ADODB.Connection")
%>
</BODY>
</HTML>
```


Pantalla 5. Objeto Session

Definición

Una **sesión** es una instancia de un **objeto que se crea en el servidor** cada vez que un **usuario entra por primera vez** a un **directorio** home o **virtual** que contenga páginas ASP.

Algunas Aclaraciones

El **objeto Session permanece hasta** que pase un determinado **periodo de tiempo** (20 minutos por defecto en Internet Information Server) **sin que el usuario realice ninguna nueva petición o** hasta que se **destruya** la **sesión** de forma explícita **mediante código**.

La finalidad de este objeto es la **gestión de una sesión de usuario** dentro del servidor. Esto se consigue mediante detección de ciertos **eventos** usando el archivo **Global.asa**, así como con la **compartición de variables entre las distintas páginas** durante el tiempo que ésta permanezca activa.

Session.Timeout: Tiempo de espera **máximo de inactividad** antes de dar por cerrada una **sesión**. Se mide en **minutos**.

Session.Abandon: **Destruye** de forma explícita una **sesión**, con todos los datos que ésta tenga asociados.

Como parte de las características que aporta ASP para la gestión de una sesión se encuentran también las denominadas **variables de sesión**. La principal diferencia con una variable normal es su ámbito o alcance. En este caso, una variable de sesión estará **disponible o visible en cualquiera de las páginas '.asp' del directorio durante el tiempo que dure la sesión**. Su principal **finalidad**, será por tanto, la **compartición de información** a lo largo de las páginas durante el tiempo que el usuario se encuentre navegando por el sitio web.

La otra característica fundamental de **estas variables** radica en que dicha información compartida es **local a cada usuario** durante toda la sesión. Es decir, cualquier **modificación sólo es visible por el usuario** cuya sesión activó la creación de la variable. Ningún otro puede ver los cambios.

Más Info:

Una **variable de sesión** es una variable que **puede verse** o está disponible **en todas las páginas '.asp' del directorio virtual y** que es **local a cada usuario**. Cuando un **usuario** entra en el directorio **se crea una copia local** de esa variable para él.

Para **crear una variable de sesión** basta poner: **Session("nombrevariable")**. Desde ese momento, **cualquier referencia posterior a la variable** debe llevar siempre el prefijo **Session("...")**.

Pantalla 6. Objeto Application

Este **objeto integrado Application** permite **manipular** ciertos aspectos dentro de lo que en ASP se conoce como **Aplicación**.

Definición

Una **aplicación** es el **conjunto de páginas '.asp'** que se encuentran dentro de un **directorio virtual**.

Algunas Aclaraciones

El **objeto Application** cuenta con los siguientes **métodos** destacados:

Application.Lock: Actúa como **semáforo**, **impidiendo** que más de un **usuario acceda de forma simultánea** a cualquier propiedad del objeto application.

Application.Unlock: **Desbloquea** el objeto Application. Es decir, **levanta el semáforo** impuesto con el método Lock.

De forma similar al objeto Session, **cuenta con** ciertos **eventos** dentro del archivo **Global.asa** y es capaz de **guardar información** de forma **global** mediante un tipo especial de variables denominadas **variables de aplicación**.

Mas info:

Una variable de aplicación es una variable que está disponible o visible en todas las páginas '.asp' del directorio virtual y que es compartida a la vez por todos los usuarios conectados al directorio. Si alguno hace cambiar el valor de esta variable, el resto percibe que ha sido modificada.

Para **crear una variable de aplicación** debe emplearse la instrucción: **Application("nombrevariable")**.

Desde ese momento, **cualquier referencia a la variable** debe llevar siempre el prefijo **Application("...")**.

Ejemplo:

Un ejemplo de variable de Aplicación y del empleo de semáforos:

```
<HTML>
<HEAD>
<TITLE>Ejemplo con Objeto Application</TITLE>
</HEAD>
<BODY>
```

```
<% ' Actualización de una variable de aplicación y
```

"Diseño de Páginas Web" - HTML Avanzado

' empleando un semáforo para la modificación.

```
Application.Lock
  Application("contador")=Application("contador")+1
Application.Unlock()
%>
Valor del contador:<%=Application("contador") %>
```

```
</BODY>
</HTML>
```

PANTALLA 7. FICHEROS. Objeto FileSystemObject

Para el **manejo de ficheros**, ASP incorpora **dos componentes externos** (objetos ActiveX), que añaden dicha funcionalidad al entorno.

- ? Un objeto de tipo **FileSystemObject**, con el que tan **sólo** se pueden **crear o abrir ficheros**.
- ? Un objeto **TextStream**, que permite, aparte del **cierre** de ficheros, la **escritura y la lectura** de los mismos.

Para **emplear estos objetos**, puesto que no son Objetos Integrados, se debe recurrir al método **CreateObject** de Server.

El método **Server.CreateObject** permite **crear instancias de objetos** que no son integrados.

Para **asignar a una variable un objeto** debe emplearse la palabra clave **Set** precediendo al nombre de la variable:

```
Set nombrevariable = Server.CreateObject("tipoObjeto")
```

El objeto con el que inicialmente podemos acceder a la manipulación de ficheros es de tipo **FileSystemObject**. por lo que para **obtener una instancia** del mismo, bastaría hacer:

```
Set f = Server.CreateObject("Scripting.FileSystemObject")
```

Los **métodos** disponibles para este objeto son:

Los **métodos** disponibles para este objeto son:

OpenTextFile: Crea o abre un archivo de texto. Devuelve un objeto de tipo **TextStream**

CreateTextFile: Crea un archivo de texto. Devuelve un objeto de tipo **TextStream**

Tanto uno como otro **devuelven** una instancia de un **objeto TextStream** con el que ya es posible realizar **operaciones de lectura y escritura** sobre el fichero.

Mas info:

El método **OpenTextFile** presenta los siguientes argumentos, los tres últimos opcionales. La llamada devuelve un objeto, por lo que se ha de recoger el valor en una variable empleando un **Set**:

```
Set text= f.OpenTextFile(ruta,modo,flagCreacion,formato)
```

ruta	Ruta absoluta del archivo que se quiere abrir o crear.
modo	Modo en que se quiere abrir el archivo: 1:lectura (por defecto). 2:escritura. 3:añadir.

flagCreacion	Si este parámetro esta a True y el fichero que se está intentando abrir no existe , entonces crea ese fichero . En caso contrario, si el fichero que se está intentando abrir no existe y este parámetro está a False (por defecto), emite un mensaje de error .
formato	False (por defecto), indica que el formato del archivo es texto ASCII . True indicará UNICODE .

El método CreateTextFile presenta tres parámetros, y devuelve también un objeto de tipo TextStream:

Set text= f.CreateTextFile(ruta,flagSobrescritura,formato)

ruta	Ruta absoluta del archivo que se quiere crear.
flagSobreescritura	Si este parámetro esta a True y el fichero que se está intentando crear ya existe , entonces sobreescribe ese fichero . En caso contrario, si el fichero que se está intentando crear ya existe y este parámetro está a False (por defecto), emite un mensaje de error .
formato	False (por defecto), indica que el formato del archivo es texto ASCII . True indicará UNICODE .

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Ejemplo manejo de Ficheros</TITLE>
</HEAD>
<BODY>
<% Set f=Server.CreateObject("Scripting.FileSystemObject")
' Creamos un fichero en c:\inetpub\scripts\ llamado fichero.txt
' poniendo el tercer parámetro a true

Set fich=f.OpenTextFile("C:\inetpub\scripts\fichero.txt",2,true)
%>
</BODY>
</HTML>
```

Pantalla 8. Ficheros. Objeto TextStream

El segundo de los objetos que se incorpora en **ASP para el manejo de ficheros** es de tipo **TextStream**. Este objeto **se obtiene siempre tras** las llamadas a **OpenTextFile** ó **CreateTextFile** de FileSystemObject. Lo único que se requiere es **almacenarlo en una variable** (instancia) para poder emplear sus métodos y propiedades:

```
Set f = Server.CreateObject("Scripting.FileSystemObject")
```

```
Set fich = f.OpenTextFile ("c:\inetpub\scripts\fichero.txt",2,true)
```

Los **métodos y propiedades** disponibles, los vamos a dividir en lectura, escritura y cerrado del fichero y siempre teniendo en cuenta que tanto la **lectura como la escritura son secuenciales y hacia adelante**.

? Métodos y propiedades de lectura:

Readline	Devuelve una línea a partir de la posición actual del puntero del fichero.
Read(n)	Devuelve tantos caracteres a partir de la posición actual del puntero como se le indique en el argumento.
ReadAll	Devuelve una cadena de texto, con todo el contenido del fichero .
Skip(n)	Ignora tantos a caracteres a partir de la posición actual del puntero como se le indique en el argumento.
Skipline	Ignora la línea a partir de la posición actual del cursor.
AtEndOfStream	Devuelve True si el puntero se encuentra en el final del fichero y False en el caso contrario.
CreateTextFile	Devuelve True si el puntero se encuentra en el final de una línea o False en caso contrario.

? Entre los **métodos de escritura** se encuentran:

Writeline(cadena)	Escribe la cadena que se pase como argumento en la posición actual del cursor e introduce un retorno de carro a continuación, es decir, lo siguiente que se escriba, se hará en la siguiente línea.
Write(cadena)	Escribe la cadena que se pase como argumento en la posición actual del cursor sin introducir a continuación ningún caracter extra . Esto es, lo siguiente que se escriba en el fichero aparecerá a continuación.
WriteBlankLines(n)	Escribe tantas líneas en blanco como se le indiquen en el argumento.

? Finalmente, existe un método que permite **cerrar el fichero**:

Close	Cierra el fichero.
--------------	--------------------

Ejemplo:

1. El objetivo será mostrar por la página Web el contenido del fichero línea a línea:

```
<HTML>
<HEAD>
<TITLE>Ejemplo lectura de ficheros</TITLE>
</HEAD>
<BODY>
<% Obtención de una instancia del objeto FileSystemObject

Set f=Server.CreateObject("Scripting.FileSystemObject")
' Obtención de la ruta física del archivo a crear. La llamada devolvería
' c:\inetpub\scripts\ llamado fichero.txt

ruta=Server.Mappath("/Scripts/ejemplos/fichero.txt")
' Abrimos para lectura
fich.OpenTextFile(ruta,1,false)
' Leemos y mostramos línea a línea el contenido
' del archivo hasta su fin
do while not fich.AtEndOfStream
    linea=fich.Readline
    response.write(linea)
    response.write("<br>")
loop

fich.close
%>
</BODY>
</HTML>
```

- 2.

```
<HTML>
<HEAD>
<TITLE>Ejemplo escritura en ficheros</TITLE>
</HEAD>
<BODY>
<% Obtención de una instancia del objeto FileSystemObject

Set f=Server.CreateObject("Scripting.FileSystemObject")
' Obtención de la ruta física del archivo a crear. La llamada devolvería
' c:\inetpub\scripts\ llamado fichero.txt

ruta=Server.Mappath("/Scripts/ejemplos/fichero.txt")
' creamos (tercer parámetro a true),
' si el fichero existiera en esa ubicación.
' Tan solo se realiza una apertura para escritura.
' Almacenamos en fich la instancia TextStream devuelta.
Set fich=f.OpenTextFile(ruta,2,true)
' escritura de una frase
fich.writeline("Esto es un ejemplo de escritura en ficheros.")

fich.close
%>
```

```
</BODY>  
</HTML>
```


Pantalla 9. RESUMEN

Al finalizar esta unidad habrás aprendido que:

1. **ASP** cuenta con la **presencia de objetos**, algunos de ellos **integrados**, que a diferencia de los objetos convencionales **pueden usarse directamente en el código** sin necesidad de ningún proceso de instanciación.
2. El **objeto integrado Request** recupera **información** proveniente **del usuario**. En función del origen de los datos, **los métodos y propiedades** se han agrupado en **colecciones**. Las más importantes son **Form, QueryString y ServerVariables**.
3. El **objeto integrado Response** se utiliza para enviar algún tipo de **salida al usuario**.
4. El **objeto integrado Server** controla parámetros del entorno de ejecución del servidor de Web.
5. El **objeto integrado Session** permite **manipular sesiones** de usuario.
6. El **objeto integrado Application** se utiliza para manipular aplicaciones.
7. Para el manejo de ficheros se emplean en ASP dos objetos. Uno de tipo **FileSystemObject** con el que solamente se puede **crear o abrir un fichero** y otro de tipo **TextStream** con el que únicamente se puede **leer, escribir o cerrar el fichero**. Ambos deben crearse empleando el método **CreateObject** del objeto integrado **Server**.
8. Cuando una **variable va a contener un objeto**, debe realizarse la **asignación mediante** la palabra clave **Set**.
9. Tanto el método **OpenTextFile** como el **CreateTextFile devuelven** un objeto **Textstream**
El objeto **TextStream** permite realizar operaciones de **lectura, escritura y cierre** de ficheros.

Módulo: ASP

Ud3. Archivo Global.asa y Acceso a bases de datos

Pantalla Objetivos formativos:

Los **objetivos formativos** que aprenderemos a lo largo de esta unidad son los siguientes:

- Conocer la estructura y finalidad de un fichero con una significación muy especial dentro de ASP. Se trata del archivo **global.asa**, que permite la detección automática de ciertos eventos dentro del servidor para poder así reaccionar en consecuencia dentro de la página ASP.
- Aprender las formas de acceder a una base de datos, mediante una conexión **ODBC**.
- Saber manipular una base de datos mediante el objeto **ADODB.Connection**.
- Conocer como almacenar una colección de registros en un objeto **recordset**. Saber cuantos **tipos** de recordset existen en Asp, así como los **métodos** de uso más importantes de estos recordset.

Pantalla 1 : Archivo Global.asa

En ASP existe un **archivo especial**, con un nombre fijo, **Global.asa**, que tiene un cometido muy específico **dentro de** lo que es un **directorio virtual**. **Sólo** puede haber **un fichero con este nombre** por aplicación, esto es, **por directorio virtual** en el servidor de web y debe estar **situado** de forma obligatoria en el **directorio raíz** de la misma.

Importante

El archivo **Global.asa** permite la detección de **eventos relacionados con sesiones y aplicaciones** así como la especificación de **scripts** que den respuesta a los mismos.

Algunas Aclaraciones

Se trata de un archivo con un **formato predeterminado**. Se encuentra compuesto por alguna **combinación de** los siguientes elementos:

- ? Eventos de Sesión.
- ? Eventos de Aplicación.

Todas las secuencias de comandos especificadas en este fichero deben ir enmarcadas entre las etiquetas de:

```
<script language="lenguajedelasecuencia" RunAt="server"> y </script>
```

La **estructura característica** de este archivo podría tener un aspecto similar al siguiente:

```
<script language="VbScript" RunAt="Server">
```

```
Sub Application_OnStart  
    ' Código de este procedimiento  
End Sub
```

```
Sub Session_OnStart  
    ' Código de este procedimiento  
End Sub
```

```
Sub Session_OnEnd  
    ' Código de este procedimiento  
End Sub
```

```
Sub Application_OnEnd  
    ' Código de este procedimiento  
End Sub
```

```
</script>
```

- ? El procedimiento **Application_OnStart** se ejecuta cuando comienza una aplicación. Esto sucede cuando **el primero de todos los usuarios**, una vez **arrancado el servidor**, pide **una cualquiera de las páginas '.asp'** que conforman la aplicación

(una cualquiera de las páginas '.asp' del directorio virtual o sus posibles subdirectorios).

- ? Las sentencias especificadas dentro de **Session_OnStart** se llevan a cabo cuando un **usuario inicia una sesión en el servidor** y pide **una cualquiera de las páginas '.asp' del directorio virtual** que constituye la aplicación. **El primer usuario de todos** que llega al servidor y solicita cualquier página '.asp' de la aplicación ejecuta, **primero el Application_OnStart**, y **después el Session_OnStart**. El resto, tan sólo lanzan el **Session_OnStart**.
- ? El procedimiento **Session_OnEnd** se ejecuta cuando **el usuario termina la sesión**, ya sea de **forma explícita** (mediante el método **Session.Abandon**) o porque **transcurra el tiempo prefijado de inactividad** que tiene el servidor para dar por terminada una sesión.
- ? La rutina **Application_OnEnd** se realiza cuando la **aplicación termina**. Esto puede suceder cuando el **servidor se para o** la máquina que lo contiene **se apaga**.

Pantalla 2: Acceso a datos mediante una conexión Odbc

Para el **acceso a bases de datos a través de web** empleando la tecnología Microsoft, se cuentan actualmente con diferentes alternativas. La más empleada hasta el momento es el **ODBC (Open DataBase Connectivity)**.

Importante

El estándar **ODBC** es el **estándar de acceso a fuentes de datos de** todos los sistemas operativos de **Microsoft**.

Algunas Aclaraciones

Una de las principales ventajas con las que cuenta este sistema consiste en que **las aplicaciones no tienen que preocuparse por el motor de la base de datos** al que tienen que acceder, sino que **con una conexión abierta** pueden **emplear tanto SQL como objetos (ADO)** para la manipulación de los datos.

La utilización de este sistema desde una página ASP permite **dos alternativas**, que tanto una como otra, deben ser **realizadas antes de cualquier manipulación** de la base de datos **desde el código** de la página:

- ? **Dar de alta las bases de datos** que se vayan a emplear en el **ODBC del sistema (DSN de Sistema)**.

- ? **Especificar el proveedor de acceso** de la base de datos **mediante código directamente en la página '.asp'**, sin tener que tocar el ODBC del panel de control para tal fin. Para ello se emplea una **cadena de texto, indicando ciertos parámetros** indispensables para efectuar la conexión. Para Microsoft Access y SQL Server estas cadenas, llamadas también **Cadenas de Conexión**, tienen la siguiente estructura:

Microsoft Access: "**DRIVER**={Microsoft Access Driver (*.mdb)}; **DBQ**=ruta física de acceso al archivo mdb; **UID**=usuario; **PWD**=password"

Microsoft SQL Server: "**DRIVER**={SQL Server}; **SERVER**=IP del servidor; **DATABASE**=nombrebasedatos; **UID**=usuario; **PWD**=password"

EJEMPLOS

Ejemplo de conexión a una base de datos Access

**ObjetoDeConexion.Open "DRIVER={Microsoft Access Driver};
DBQ=c:\inetpub\scripts\base.mdb"**

Ejemplo de conexión a una base de datos Sql Server

**ObjetoDeConexion.Open "DRIVER={SQL Server}; SERVER=
192.168.0.2; DATABASE=datos; UID=atrujillo; PWD=atrujillo"**

Pantalla 3: Objeto de Conexión

Para **manipular una base de datos** a través de una página Web, **ASP** proporciona un objeto no integrado, **ADODB.Connection**, que permite la **conexión y ejecución de consultas**.

Como todo objeto no integrado, se requiere un proceso de instanciación para poder emplear los métodos y propiedades que trae consigo.

Importante:

Para **instanciar un objeto ADODB.Connection** basta la sentencia:
Set conexion = Server.CreateObject("ADODB.Connection")

Algunas Aclaraciones:

Los **métodos más significativos** de este objeto se presentan a continuación:

Open (cadena): Abre una conexión con una base de datos. La **cadena** que se le pasa como **argumento**, debe ser o bien un nombre de **DSN de Sistema** o bien una **Cadena de Conexión**.

Execute(consulta): Ejecuta la consulta que se le pase como argumento. La consulta debe ser una **cadena de texto con la sentencia SQL** que se quiere enviar a la base de datos.

Close: Cierra la conexión.

Mas Info:

Cuando la **consulta SQL** que se manda a la base de datos **devuelve una colección de registros** (como es el caso de las consultas SELECT o de selección), **se devuelve tras la ejecución un objeto especial**, denominado **RECORDSET**, que debe **almacenarse en una variable** (instancia), para poder manipular así estos resultados.

Ejemplo

El siguiente ejemplo, muestra las acciones a seguir para almacenar los resultados de una sentencia SELECT.

```
<HTML>
<HEAD>
<TITLE>Ejemplo Acceso a Datos</TITLE>
</HEAD>
<BODY>
```

```
<% ' Obtención de una instancia del objeto Connection
Set bd=Server.CreateObject("ADODB.Connection")
' Apertura de una conexión a una base de datos registrada
' en el DSN como basedatos
```

```
bd.Open "basedatos"
' Generación y ejecución de una consulta y
' Asignación de los resultados a un Recordset
```

"Diseño de Páginas Web" - HTML Avanzado

```
    consulta="Select * from Tabla1"
    Set resultado=bd.Execute(consulta)
    ' Cerrar la conexión
    bd.Close
%>

</BODY>
</HTML>
```

PANTALLA 4. Objeto Recordset

Importante.

El **objeto Recordset** es un objeto destinado a **almacenar una colección de registros**, obtenida **tras la ejecución de una consulta** en la base de datos.

Algunas Aclaraciones.

En ASP existen **cuatro tipos** de Recordset:

adOpenForwardOnly: Tiene asociado el **valor 0**. Es el que se crea por defecto. Recorre los registros secuencialmente y sólo hacia adelante. No permite la modificación de registros. No permite ver los registros modificados, añadidos o borrados por otros usuarios. Es el de **mejor rendimiento** cuando solo se realiza una única pasada sobre el recordset.

adOpenKeySet: Tiene asociado el **valor 1**. Permite **movernos tanto hacia delante como hacia atrás** en el recordset. Se pueden **ver los cambios realizados por otros usuarios a excepción de las altas**.

adOpenDynamic: Tiene asociado el **valor 2**. Permite **movernos hacia atrás y hacia adelante** en el recordset. Se pueden **ver todos los cambios realizados por otros usuarios**.

adOpenStatic: Tiene asociado el **valor 3**. Permite el **movimiento en los dos sentidos**. No es posible ver las **modificaciones** realizadas en los registros.

Pantalla 5. Registro tipo `adOpenForwardOnly`

Por defecto y si no se indica nada más, el recordset con el que se trabaja por defecto es el de **tipo 0 (`adOpenForwardOnly`)**.

Las **propiedades y métodos** para este **Recordset `adOpenForwardOnly`** son:

EOF: Devuelve true si el puntero que marca el registro actual dentro del Recordset se encuentra después del último.

MoveNext: Avanza el puntero o cursor al siguiente registro en el Recordset.

Una vez almacenados los registros que devuelve la consulta en el objeto Recordset, se debe proceder a **manipular los datos** de cada uno ellos, **accediendo a cada campo** que compone el **registro**.

Para acceder a cada campo de un registro almacenado dentro de un Recordset, habría que ejecutar la siguiente sentencia: **variable-que-contiene-el-Recordset("campo-del-registro")**. Por ejemplo para acceder al campo apellido de una variable recordset denominada resultado que contiene todos los registros, usaríamos la sentencia:

```
Resultado("apellido")
```

Tras la ejecución de una consulta, el **cursor** del recordset se encuentra **situado en el primer registro** que se obtiene como resultado.

Una vez, se haya terminado de trabajar con el registro actual, se debe **avanzar el cursor al siguiente registro del Recordset**. Para ello se emplea el método **movenext**.

Masinfo:

La manipulación de los resultados obtenidos tras una consulta debe continuar hasta que **ya no queden registros en el Recordset**, hecho que puede controlarse con la propiedad **EOF**, que devuelve **true** si el **cursor se encuentra más allá del último registro** del Recordset.

A continuación se muestra un ejemplo, que imprime en la página Web todos los registros de una tabla que está compuesta por registros de tres campos (nombre, apellido, direccion):

EJEMPLO:

A continuación se muestra un ejemplo, que imprime en la página Web todos los registros de una tabla que está compuesta por registros de tres campos (nombre, apellido, direccion):

```
<HTML>
<HEAD>
<TITLE>Ejemplo 2 Acceso a Datos</TITLE>
</HEAD>
<BODY>

<% ' Obtención de una instancia del objeto Connection
Set bd=Server.CreateObject("ADODB.Connection")
' Apertura de una conexión a una base de datos registrada
' en el DSN como basedatos
```

```
bd.Open "basedatos"
' Generación y ejecución de una consulta y
' Asignación de los resultados a un Recordset
consulta="Select * from Tabla1"
Set resultado=bd.Execute(consulta)
' Muestra los resultados en la página web.
do while not resultado.EOF %>
    <%= resultado("nombre") %> &nbsp;
    <%= resultado("apellido") %> &nbsp;
    <%= resultado("direccion") %> <br>
    <% resultado.movenext
loop
' Cerrar la conexión
bd.Close
%>

</BODY>
</HTML>
```

Pantalla 6. Otros tipos de registros

El **Recordset por defecto**, es un cursor **muy eficiente y debe ser utilizado cuando sea posible**. Sin embargo, **existen muchas otras operaciones** disponibles sobre un conjunto de registros obtenidos en una consulta y que **no son admitidas por un Recordset adOpenForwardOnly**.

Para **trabajar por un Recordset** distinto del que se tiene por defecto, habría que seguir los siguientes **pasos**:

- ? **Crear un objeto de Conexión** a la base de datos (ADODB.Connection).
- ? **Conectar** con la base de datos.
- ? **Crear un objeto Recordset** (ADODB.Recordset).
- ? **Establecer el tipo** de Recordset.
- ? **Vincular el Recordset con una conexión abierta** a una base de datos (ActiveConnection).
- ? **Generar la consulta** en forma de cadena de texto.
- ? **Ejecutar la consulta usando** el objeto **Recordset** (método **open** del Recordset).
- ? **Recorrer los resultados** almacenados en el Recordset tras la ejecución.

Más Info:

En general, **las propiedades y métodos de un Recordset dependen de su tipo**.

Principalmente se encuentran:

EOF	Devuelve true si el puntero que marca el registro actual dentro del Recordset se encuentra después último .
MoveNext	Avanza el puntero o cursor al siguiente registro en el Recordset.
ActiveConnection	Vincula el Recordset con una Conexión de base de datos (objeto ADODB.Connection)
BOF	Devuelve true si el puntero que marca el registro actual dentro del Recordset se encuentra antes del primer registro
Open(consulta)	Ejecuta la consulta que se pasa como argumento y almacena los registros del resultado en el recordset que esta haciendo esta llamada .
Move n	Avanza ese número de registros en el Recordset . Un número negativo, retrocede en el Recordset. (Soportado para los tipos 1,3. Para tipo 2, depende del proveedor).
MovePrevious	Retrocede el puntero al registro anterior en el Recordset.
MoveLast	Mueve el puntero del recordset al último registro . El recordset debe soportar bookmark y movimiento hacia atrás. Es decir, tipos 1 y 3. Para el 2 depende del proveedor.
MoveFirst	Sitúa el puntero del recordset en el primer registro .
RecordCount	Devuelve el número de registros que conforman el Recordset . Si este valor es -1, no se ha podido determinar el número, o el Recordset no soporta el método. Los Recordsets de tipos 1 y 2 admiten esta propiedad. Para el tipo 3, depende de la fuente de datos. Para el tipo 0 no se admite.
CursorType	Establece el Recordset al tipo indicado: 0 : adOpenForwardOnly 1 : adOpenKeyset 2 : adOpenDynamic 3 : adOpenStatic
PageSize	Indica el número de registros que conformarán una página . (Cuando se

	trabaja con paginación en el Recordset). Por defecto,10.
PageCount	Devuelve el número de páginas en las que se ha estructurado el Recordset . (-1, si el Recordset no soporta el método).
AbsolutePage	Establece el cursor del Recordset en el número de página que se indique . (Cuando se trabaja con paginación en el Recordset).

EJEMPLO

```
<HTML><HEAD>
<TITLE>Ejemplo 3 Acceso a Datos</TITLE>
</HEAD><BODY>

<% ' Obtención de una instancia del objeto Connection
Set bd=Server.CreateObject("ADODB.Connection")
' Apertura de una conexión a una base de datos registrada
' en el DSN como basedatos

bd.Open "basedatos"

' Obtención de una instancia del objeto Recordset
Set resultado=Server.CreateObject("ADODB.Recordset")
' Establecer el Recordset que se quiera emplear.
' En nuestro caso adOpenKeySet
resultado.CursorType=1
' Vinculación del Recordset con una base de datos abierta.
resultado.ActiveConnection=bd

' Generación y ejecución de una consulta. Asignación de los resultados a un recordset

consulta="Select * from Tabla1"
resultado.Open consulta
' contar el número de registros del resultado.
total=resultado.RecordCount
response.write(" se han encontrado " & total & " registros <br>")
' Muestra los resultados en la página web.
do while not resultado.EOF %>
    response.write(resultado("nombre")) & "&nbsp;"
    response.write(resultado("apellido")) & "&nbsp;"
    response.write(resultado("direccion")) & "&nbsp;"
    resultado.movenext
loop
' Cerrar la conexión
bd.Close
%>
</BODY></HTML>
```

RESUMEN

El archivo **Global.asa** detecta ciertos **eventos de sesión y aplicación dentro del directorio virtual** donde residen las páginas ASP de la aplicación. Tiene una **estructura fija** formada por los **procedimientos** que deben ejecutarse como respuesta a dichos eventos.

ODBC permite la conexión a fuentes de datos del sistema para poder trabajar así con ellas desde las aplicaciones. Desde ASP las **dos alternativas** para manipular el **ODBC**, son el **DSN de Sistema** (localizado en las Fuentes de Datos ODBC del panel de Control del Servidor) **y las Cadenas de Conexión**.

El objeto **ADODB.Connection** permite **abrir y cerrar conexiones** así como **ejecutar consultas** en una Base de Datos.

Cuando la ejecución de la consulta SQL devuelve una colección de **registros**, el **método execute devuelve** un **objeto RECORDSET** con dichos registros.

El **objeto Recordset** almacena una **colección de registros** obtenidos **tras** la ejecución de una **consulta**.

El **Recordset por defecto es el tipo 0**, también denominado **adOpenForwardOnly**, que tan **sólo permite el movimiento hacia adelante** dentro del conjunto de registros que se encuentran almacenados. Además de este tipo de recordset existen otros tres tipos más.

Los **Recordsets** cuentan con **diversas funciones para el manejo de los resultados** de una consulta.

Unidad Didáctica: Hojas de Estilo en Cascada (CSS)

Índice

Pantalla: 1. Objetivos formativos

- ? Conocer las posibilidades que nos ofrecen las hojas de estilo en cascada (CSS)
- ? Saber crear estilos personalizados, redefinir etiquetas HTML y emplear los selectores CSS.
- ? Saber aplicar estilos a una página Web desde dentro del propio documento o de forma vinculada.
- ? Saber discriminar en qué casos se aplicará un estilo a una sección del documento Web o al documento completo.

Pantalla: 2. Primera aproximación al CSS

Definición:

Un estilo es un grupo de atributos de formato que controla la apariencia de un rango de texto en un documento o bien del documento completo.

Desarrollo:

Hay tres tipos de hojas de estilos CSS:

- ? **Los estilos CSS personalizados** son similares a los que se emplean en los programas de tratamiento de texto, salvo que no distinguen entre estilo de carácter y de párrafo. Puede aplicar estilos CSS personalizados a cualquier rango o bloque de texto.
- ? **Los estilos de etiquetas HTML** redefinen el formato de una determinada etiqueta, como h1. Cuando se crea o cambia un estilo CSS para la etiqueta h1, todo el texto formateado con la etiqueta h1 se actualiza inmediatamente.
- ? **Los estilos del selector CSS.** Son estilos diseñados específicamente para manipular las características visuales de los hiperenlaces en función de su estado.

Masinfo:

Los estilos CSS existen desde hace algún tiempo, pero muchos diseñadores y creadores de sitios Web se han mostrado reacios a usarlos debido a que no siempre son compatibles con los navegadores. Sin embargo, si los usuarios del sitio utilizan navegadores que reconocen los estilos CSS (**versión 4.0 y posteriores**), debería aprovechar la potencia y el control que ofrecen.

Pantalla: 3. Características y ventajas del CSS

Desarrollo:

El modo de funcionamiento de las CSS consiste en definir, mediante una sintaxis especial, la forma de presentación que le aplicaremos a:

- ? Un Web entero, de modo que se puede definir la forma de todo el Web de una sola vez.
- ? Un documento HTML o página, se puede definir la forma, en un pequeño trozo de código en la cabecera, a toda la página.
- ? Una parte del documento, aplicando estilos visibles en una sección de la página.
- ? Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta.

Además esta sintaxis CSS permite aplicar al documento formato de modo mucho más exacto:

- ? Podemos definir la distancia entre líneas del documento.
- ? Se puede aplicar indentado a las primeras líneas del párrafo.
- ? Podemos colocar elementos en la página con mayor precisión, y sin lugar a errores.
- ? Y mucho más, como definir la visibilidad de los elementos, márgenes, subrayados, tachados...

Si con el HTML tan sólo podíamos definir atributos en las páginas con pixeles y porcentajes, ahora podemos definir utilizando muchas más unidades como:

- ? Pixels (px) y porcentaje (%), como antes.
- ? Pulgadas (in)
- ? Puntos (pt)
- ? Centímetros (cm)

Pantalla: 4. Sintaxis CSS

Introducción:

Tal como y como veremos en los ejemplos la sintaxis es bastante sencilla y repetitiva.

Desarrollo:

- ? Para definir un estilo se utilizan atributos como font-size, text-decoration... seguidos de dos puntos y el valor que le deseemos asignar. Podemos definir un estilo a base de definir muchos atributos separados por punto y coma.

Ejemplo:

font-size: 10pt; text-decoration: underline; color: black; (el último punto y coma de la lista de atributos es opcional)

- ? Para definir el estilo de una etiqueta se escribe la etiqueta seguida de la lista de atributos encerrados entre llaves.

Ejemplo:

H1{text-align: center; color: black}

- ? Los valores que se pueden asignar a los atributos de estilo se pueden ver en una tabla en el siguiente capítulo. Muchos estos valores son unidades de medida, por ejemplo, el valor del tamaño de un margen o el tamaño de la fuente. Las unidades de medida son las siguientes:

Puntos	pt
Pulgadas	in
Centímetros	cm
pixels	px

Pantalla: 5. Atributos de las hojas de estilo

Desarrollo:

Proporcionamos a continuación una clasificación de los distintos atributos que pueden aplicarse a los documentos HTML y los valores que pueden adoptar:

FUENTES - FONT		
color	valor RGB o nombre de color	color: #009900; color: red;
Sirve para indicar el color del texto. Lo admiten casi todas las etiquetas de HTML. No todos los nombres de colores son admitidos en el estándar, es aconsejable entonces utilizar el valor RGB.		
font-size	xx-small x-small small medium large x-large xx-large Unidades de CSS	font-size: 12pt; font-size: x-large;
Sirve para indicar el tamaño de las fuentes de manera más rígida y con mayor exactitud.		
font-family	serif sans-serif cursive fantasy monospace Todas las fuentes habituales	font-family: arial, helvetica; font-size: fantasy;
Con este atributo indicamos la familia de tipografía del texto. Los primeros valores son genéricos, es decir, los exploradores los comprenden y utilizan las fuentes que el usuario tenga en su sistema. También se pueden definir con tipografías normales, como ocurría en html. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien.		
font-weight	normal bold bolder lighter 100 200 300 400 500 600 700 800 900	font-weight: bold; font-weight: 200;
Sirve para definir la anchura de los caracteres, o dicho de otra manera, para poner negrillas con total libertad. Normal y 400 son el mismo valor, así como bold y 700.		
font-style	normal italic oblique	font-style: normal; font-style: italic;
Es el estilo de la fuente, que puede ser normal, itálica u oblicua. El estilo oblique es similar al italic.		

PÁRRAFOS - TEXT		
line-height	normal y unidades CSS	line-height: 12px; line-height: normal;
El alto de una línea, y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando HTML.		
text-decoration	none [underline overline line-through]	text-decoration: none; text-decoration: underline;
Para establecer la decoración de un texto, es decir, si está subrayado, sobrerayado o tachado.		
text-align	left right center justify	text-align: right; text-align: center;
Sirve para indicar la alineación del texto. Es interesante destacar que las hojas de estilo permiten el justificado de texto, aunque recuerda que no tiene por qué funcionar en todos los sistemas.		
text-indent	Unidades CSS	text-indent: 10px; text-indent: 2in;
Un atributo que sirve para hacer sangrado o márgenes en las páginas. Muy útil y novedosa.		
text-transform	capitalize uppercase lowercase none	text-transform: none; text-transform: capitalize;
Nos permite transformar el texto, haciendo que tenga la primera letra en mayúsculas de todas las palabras, todo en mayúsculas o minúsculas.		

FONDO - BACKGROUND		
Background-color	Un color, con su nombre o su valor RGB	background-color: green; background-color: #000055;

Sirve para indicar el color de fondo de un elemento de la página.

Background-image	El nombre de la imagen con su camino relativo o absoluto	background-image: url(mármol.gif) ; background-image: url(http://www.x.com/fondo.gif)
-------------------------	--	--

Colocamos con este atributo una imagen de fondo en cualquier elemento de la página.

BOX - CAJA		
Margin-left	Unidades CSS	margin-left: 1cm; margin-left: 0,5in;

Indicamos con este atributo el tamaño del margen a la izquierda

Margin-right	Unidades CSS	margin-right: 5%; margin-right: 1in;
---------------------	--------------	---

Se utiliza para definir el tamaño del margen a la derecha

Margin-top	Unidades CSS	margin-top: 0px; margin-top: 10px;
-------------------	--------------	---------------------------------------

Indicamos con este atributo el tamaño del margen arriba de la página

Margin-bottom	Unidades CSS	margin-bottom: 0pt; margin-top: 1px;
----------------------	--------------	---

Con el se indica el tamaño del margen en la parte de abajo de la página

Border-color	color RGB y nombre de color	border-color: red; border-color: #ffccff;
---------------------	-----------------------------	--

Para indicar el color del borde del elemento de la página al que se lo aplicamos. Se puede poner colores por separado con los atributos border-top-color, border-right-color, border-bottom-color, border-left-color.

Border-style	none dotted solid double groove ridge inset outset	border-style: solid; border-style: double;
---------------------	--	---

El estilo del borde, los valores significan: none=ningun borde, dotted=punteado (no parece funcionar), solid=solido, double=doble borde, y desde groove hasta outset son bordes con varios efectos 3D.

border-width	Unidades CSS	border-width: 10px; border-width: 0.5in;
---------------------	--------------	---

El tamaño del borde del elemento al que lo aplicamos.

Para ver otros ejemplos de Box [pulsar aquí](#)

float	none left right	float: right;
--------------	---------------------	---------------

Sirve para alinear un elemento a la izquierda o la derecha haciendo que el texto se agrupe alrededor de dicho elemento. Igual que el atributo align en imágenes en sus valores right y left.

clear	none right left	clear: right;
--------------	---------------------	---------------

Si este elemento tiene a su altura imágenes u otros elementos alineados a la derecha o la izquierda, con el atributo clear hacemos que se coloquen en un lugar donde ya no tenga esos elementos a el lado que indiquemos.

Para ver una página que utiliza float y clear

Pantalla: 6. Usos del CSS (I)

Introducción:

Vamos ahora a describir los diferentes usos de las CSS introducidos anteriormente.

Desarrollo:

Pequeñas partes de la página

Para definir estilos en secciones reducidas de una página se utiliza la etiqueta ****. Con su atributo **style** indicamos en sintaxis CSS las características de estilos. Lo vemos con un ejemplo, pondremos un párrafo en el que determinadas palabras las vamos a visualizar en color verde.

```
<p>
Esto es un texto con algunas palabras <SPAN style="color:green">en color verde</SPAN>. Pero
otras no.
</p>
```

Que tiene como resultado:

Esto es un texto con algunas palabras **en color verde**. Pero otras no.

Ejemplo:

```
<HTML>
```

```
    <HEAD>
```

```
        <TITLE> Estilos CSS (I) </TITLE>
```

```
    </HEAD>
```

```
    <BODY>
```

```
        <p>
```

```
        Esto es un texto con algunas palabras <SPAN style="color:green">en
        color verde</SPAN>. Pero otras no.
```

```
        </p>
```

```
    </BODY>
```

```
</HTML>
```

Pantalla: 7. Usos del CSS (II)

Desarrollo:

Estilo definido para una etiqueta

De este modo podemos hacer que toda una etiqueta muestre un estilo determinado. Por ejemplo, podemos definir un párrafo entero en color rojo y otro en color azul. Para ello utilizamos el atributo **style**, que es admitido por todas las etiquetas del HTML (siempre y cuando dispongamos de un navegador compatible con CSS).

```
<p style="color:#990000">
Esto es un párrafo de color rojo.
</p>
<p style="color:#000099">
Esto es un párrafo de color azul.
</p>
```

Que tiene como resultado:

Esto es un párrafo de color rojo.

Esto es un párrafo de color azul.

Ejemplo:

<HTML>

<HEAD>

<TITLE> Estilos CSS (II) **</TITLE>**

</HEAD>

<BODY>

<p style="color: #990000">Esto es un párrafo de color rojo.**</p>**

<p style="color: #000099">Esto es un párrafo de color azul.**</p>**

</BODY>

</HTML>

Pantalla: 8. Usos del CSS (III)

Desarrollo:

Estilo definido en una parte de la página

Con la etiqueta **<DIV>** podemos definir secciones de una página y aplicarle estilos con el atributo **style**, es decir, podemos definir estilos de una vez a todo un bloque de la página.

```
<div style="color:#000099; font-weight:bold">
<h3>Estas etiquetas van en <i>azul y negrita</i></h3>
<p>
Seguimos dentro del DIV, luego permanecen los etilos
</p>
</div>
```

Que tiene como resultado:

[Estas etiquetas van en azul y negrita](#)

Seguimos dentro del DIV, luego permanecen los etilos

Ejemplo:

<HTML>

<HEAD>

<TITLE> Estilos CSS (II) **</TITLE>**

</HEAD>

<BODY>

```
<div style="color:#000099; font-weight:bold">
<h3>Estas etiquetas van en <i>azul y negrita</i></h3>
<p>
Seguimos dentro del DIV, luego permanecen los etilos
</p>
</div>
```

</BODY>

</HTML>

Pantalla: 9. Usos del CSS (IV)

Desarrollo:

Estilo definido para toda una página

Podemos definir, en la cabecera del documento, estilos para que sean aplicados a toda la página. Es una manera muy cómoda de darle forma al documento y muy potente, ya que estos estilos serán seguidos en toda la página y nos ahorraremos así muchas etiquetas HTML que apliquen forma al documento. Además, si deseamos cambiar los estilos de la página lo haremos de una sola vez.

En este punto vemos que se utiliza la etiqueta **<STYLE>** colocada en la cabecera de la página para definir los distintos estilos del documento.

A grandes rasgos, entre de **<STYLE>** y **</STYLE>**, se coloca el nombre de la etiqueta que queremos definir los estilos y entre llaves **{ }** colocamos en sintaxis CSS las características de estilos.

```
<STYLE type="text/css">
<!--
H1 {text-decoration: underline; text-align:center}
P {font-family:arial,verdana; color: white; background-color: black}
BODY {color:black;background-color: #cccccc; text-indent:1cm}
// -->
</STYLE>
```

Como se puede apreciar en el código, hemos definido que la etiqueta **<H1>** se presentará

- ? Subrayado
- ? Centrada

También, por ejemplo, hemos definido que el cuerpo entero de la página (etiqueta **<BODY>**) se le apliquen los estilos siguientes:

- ? Color del texto negro
- ? Color del fondo grisáceo
- ? Margen lateral de 1 centímetro

Ejemplo:

```
<html>
  <head>
    <title>Uso CSS (IV)</title>
    <STYLE type="text/css">
      H1 {
        text-decoration: underline;
        text-align:center
      }
      P {
        font-Family: arial,verdana;
        color: white;
        background-color: black
      }
      BODY {
        color:black;
        background-color: #cccccc;
        text-indent: 1cm;
      }
    </STYLE>
  </head>
  <body>
    <h1>Página con estilos incluidos en el documento</h1>

    <p>Este párrafo ya tiene definida su etiqueta</p>
  </body>
</html>
```

Masinfo:

Caber destacar que si aplicamos estilos a la etiqueta **<BODY>**, estos serán heredados por el resto de las etiquetas del documento. Esto es así siempre y cuando no se vuelvan a definir esos estilos en las siguientes etiquetas, en cuyo caso el estilo de la etiqueta más concreta será el que mande. Puede verse este detalle en la etiqueta **<P>**, que tiene definidos estilos que ya fueron definidos para **<BODY>**. Los estilos que se tienen en cuenta son los de la etiqueta **<P>**, que es más concreta.

Desarrollo:

Una de las características más potentes de la programación con hojas de estilos consiste en que, de una vez, podemos definir los estilos de todo un sitio Web. Esto se consigue creando un archivo donde tan sólo colocamos las declaraciones de estilos de la página y enlazando todas las páginas del sitio con ese archivo. De este modo, todas las páginas comparten una misma declaración de estilos y, por tanto, si la cambiamos, cambiarán todas las páginas.

Veamos ahora cómo el proceso para incluir estilos con un fichero externo:

1- Creamos el fichero con la declaración de estilos

Es un fichero de texto normal, que puede tener cualquier extensión, aunque le podemos asignar la extensión .css para aclararnos qué tipo de archivo es. El texto que debemos incluir debe ser escrito exclusivamente en sintaxis CSS.

2- Enlazamos la pána web con la hoja de estilos

Para ello, vamos a colocar la etiqueta <LINK> con los atributos

- ? **rel="STYLESHEET"** indicando que el enlace es con una hoja de estilos
- ? **type="text/css"** porque el archivo es de texto, en sintaxis CSS
- ? **href="estilos.css"** indica el nombre del fichero fuente de los estilos

Ejemplo:

Archivo .CSS	Archivo .HTML
<pre>P { font-size : 12pt; font-family : arial,helvetica; font-weight : normal; } H1 { font-size : 36pt; font-family : verdana,arial; text-decoration : underline; text-align : center; background-color : Teal; } TD { font-size : 10pt; font-family : verdana,arial; text-align : center; background-color : 666666; } BODY { background-color : #006600; font-family : arial; color : White; }</pre>	<pre><html> <title>Página HTML con estilos enlazados</title> <head> <link rel="STYLESHEET" type="text/css" href="estilos.css"> </head> <body> <h1>Texto de cabecera</h1> Esta pá&acute;gina tiene en la cabecera la etiqueta necesaria para vincular la hoja de estilos.
 <table width="300" cellspacing="2" cellpadding="2" border="0"> <tr> <td>Esto est&acute; dentro de un TD, luego tiene estilo propio, declarado en el fichero externo</td> </tr> <tr> <td>Segunda columna</td> </tr> </table> </body> </html></pre>

Desarrollo:

Se define el estilo de los enlaces asignando su apariencia en sus distintos estados:

- ? Enlace no visitado. Se define con el atributo **link**.
- ? Enlace visitado. Se define con el atributo **visited**.
- ? Enlace activo (cuando se está pulsando). Se define con **active**.
- ? Enlace con el ratón encima. Se define con **hover**.

Esta definición se realiza en la cabecera de la página, entre las etiquetas **<STYLE>** y **</STYLE>**, y es global a toda la página.

Un ejemplo de esto se puede ver en esta declaración de estilos:

```
<STYLE type="text/css">
  A:link {text-decoration: none;color: #0000cc;}
  A:visited {text-decoration: none;color: #ffcc33;}
  A:active {text-decoration: none;color: #ff0000;}
  A:hover {text-decoration: underline;color: #999999;}
</STYLE>
```

Ejemplo:

```
<html>
  <head>
    <title>Selectores CSS</title>
    <STYLE type="text/css">
      A:link {
        text-decoration: none;
        color: #0000cc;
      }
      A:visited {
        text-decoration: none;
        color: #ffcc33;
      }
      A:active {
        text-decoration: none;
        color: #ff0000;
      }
      A:hover {
        text-decoration: underline;
        color: #999999;
      }
    </STYLE>
  </head>
  <body>
    <a href="http://www.terra.es">Enlace a Terra</a>
  </body>
</html>
```

Pantalla: Resumen

Al final de esta unidad habrás aprendido:

- ? Gracias a las **Hojas de Estilo en Cascada (Cascade Style Sheet)** podrás superar muchas de las limitaciones que plantea el lenguaje HTML en cuanto a **efectos gráficos** sobre el texto y otros objetos.
- ? El CSS permite **aplicar un estilo** determinado a:
 - o Una sección del documento.
 - o Un documento completo.
 - o El sitio Web entero.
- ? Existen **tres posibilidades** a la hora de definir estilos CSS:
 - o Crear **estilos personalizados**, los cuales decidiremos donde y sobre qué elemento aplicar.
 - o **Redefinir etiquetas HTML**, con lo que no es necesario aplicar el estilo dentro del documento HTML ya que la propia etiqueta lo lleva implícito.
 - o Utilizar los **selectores CSS**, estilos específicos para los hiperenlaces que se aplican en función del estado en que se encuentren.
- ? Los estilos pueden definirse dentro **del propio documento HTML**, con lo cual solo afectarían a ese documento, puede crearse un **archivo .css** que contenga todos los estilos y afectaría a todos los documento HTML a los que se vinculase.
- ? Cuando modificamos un estilo los cambios repercuten **inmediatamente** sobre todos los objetos a los que esté aplicado, lo cual facilita el **mantenimiento y actualizaciones** de cualquier sitio Web en el que se empleen.

Módulo: Programación Java (APPLETS)

PANTALLA 1 Objetivos formativos:

Los **objetivos formativos** que aprenderemos a lo largo de esta unidad son los siguientes:

- ? Aprender que un applet es una **aplicación web** que ejecutaremos desde un navegador web
- ? Conocer cual es la forma de incluir un applet en una página web mediante el uso de las etiquetas HTML **<APPLET>** y **<PARAM>**.
- ? Saber cuales son las **restricciones** que nos podemos encontrar dependiendo del navegador y de los niveles de seguridad establecidos.
- ? Aprender a manejar las **capacidades** que nos ofrecen los applets en Java y que otras aplicaciones no tienen.
- ? Conocer los **métodos** que podemos emplear para poder usar un applet, algunos de los cuales son obligatorios y otros son opcionales.

Pantalla: 2. ¿Qué es un Applet?

Definición:

Un **applet** es una **pequeña aplicación** accesible en un servidor de Internet, que se transporta por la red, se instala automáticamente en la **máquina cliente** y es ejecuta por ésta in situ como parte de un **documento web** (página html).

Desarrollo:

Es fundamental tener claro que un **applet** es descargado desde la máquina servidora a la **cliente**, a través de una red basada en TCP/IP (Internet), y **ejecutado siempre por la máquina cliente**.

En realidad, un **applet** es una aplicación pretendidamente corta basada en un formato gráfico sin representación independiente: es decir, se trata de un elemento a embeber en otras aplicaciones; es un componente en su sentido estricto.

Un applet es una mínima aplicación Java diseñada para ejecutarse en un navegador Web. Por tanto, no es necesario la creación de un método **main()** que provoque el inicio de la ejecución del applet. Será el navegador el que se encargue de comenzar su ejecución. El applet asume que el código se está ejecutando desde dentro de un navegador.

Más info:

El hecho de que se integre estática (embebido en un ejecutable) o dinámicamente (intérpretes, DLLs, etc.) no afecta en absoluto a la esencia de su comportamiento como **componente** con el que construir diálogos con sentido autónomo. Los applets se han construido mayoritariamente, y con gran acierto comercial, como pequeñas aplicaciones interactivas, con movimiento, animaciones y sonido... en Internet.

Pantalla: 3. Clases para crear applets

Todo **applet** tiene que ser una clase que herede de la clase **java.applet.Applet**.

La superclase inmediata de **Applet** es **java.awt.Panel**, y como todas las subclases de **Panel**, **Applet** obtiene su propia área de dibujo. La superclase inmediata de **java.awt.Panel** es **java.awt.Container**, lo que le permite a **Panel** y por herencia a **Applet** tener otros componentes en su interior.

La siguiente en el árbol jerárquico de **Applet** es **java.awt.Component** (**Container** hereda de **Component**). Todos los controles gráficos de Java son subclases de **Component**, y Applet no es una excepción. Gracias a ser subclase de Component, un applet puede ser agrupado en un Container, y posicionado mediante un gestor de presentación, lo que significa que **un applet puede colocarse en cualquier Container y no sólo en un navegador**.

Pantalla: 4 Capacidades y Restricciones

Los applets tendrán una serie de restricciones dependiendo del navegador.

Restricciones

Los Navegadores actuales imponen las siguientes **restricciones** a los applets que se cargan a través de la Red:

- ? Un applet no puede cargar librerías ni definir métodos nativos.
- ? No puede leer ni escribir ficheros en el Host en el que se está ejecutando.
- ? No puede realizar conexiones en la Red, excepto con el Host del que fue cargado.
- ? No puede arrancar ningún programa en el Host donde se está ejecutando.
- ? No puede leer ciertas propiedades del sistema.

Las ventanas que proporcionan los applets tienen un aspecto diferente a las de cualquier aplicación.

Cada Navegador tiene un objeto **SecurityManager** que implementa sus controladores de seguridad. Cuando el objeto **SecurityManager** detecta una violación, lanza una **SecurityException** (Excepción de seguridad). Su applet puede capturar esta **SecurityException** y reaccionar del modo apropiado.

Y además de las restricciones, los applets tienen una serie de **capacidades**.

Capacidades

El paquete **java.applet** proporciona una API que contiene algunas **capacidades** de los applets que las aplicaciones no tienen. Por ejemplo, los applets pueden ejecutar sonidos, que otros programas no pueden todavía.

Aquí capacidades de los applets son:

- ? Los Applets pueden hacer conexiones al host del que fueron cargados.
- ? Los Applets que se ejecutan dentro de un navegador Web pueden hacer que se muestren páginas HTML de una forma muy sencilla.
- ? Los Applets pueden invocar métodos públicos de otros Applets que se encuentren en la misma página.
- ? Los Applets que se han cargado desde un directorio local (desde un directorio en el CLASSPATH del usuario) no tienen ninguna restricción como los applets cargados a través de la Red.
- ? Aunque la mayoría de los applets paran su ejecución cuando el usuario abandona la página, no tienen porque hacerlo.

Mas info:

Los controladores de Seguridad también están sujetos a cambios. Por ejemplo, si el navegador está desarrollado para trabajar solo en entornos fiables, entonces sus controladores de seguridad serán mucho más flojos que los descritos en esta página.

PANTALLA 5: Applets en la web

Dado que los applets están mayormente destinados a ejecutarse en navegadores Web, había que preparar el lenguaje HTML para soportar Java, o mejor, los applets. Para ello se añadió al HTML dos nuevas etiquetas y sus correspondientes atributos. Las etiquetas son `<APPLET>` Y `<PARAM>`.

Veamos un primer ejemplo de código de un applet. Como ya sabemos todo applet que construyamos tiene que ser una clase que herede de **Applet**. El paquete en el que se encuentran las clases relacionadas con los applets es **java.applet**. Por lo que tendremos que importar este paquete.

```
import java.applet.Applet;
import java.awt.Graphics;

public class PrimerApplet extends Applet{
    public void paint(Graphics g){
        g.drawString("¡¡¡Este es nuestro primer applet!!!",20,25);
    }
}
```

La clase **PrimerApplet** lo único que hace es reescribir el método `paint` que hereda de **Applet** y es el que se encarga de "pintar" en la zona gráfica que ocupa el applet dentro de la página web.

Para poder mostrar este applet en una página HTML tenemos que indicarlo en la página mediante la etiqueta `<APPLET>`.

```
<html>
<head>
    <title>El primer applet </title>
</head>
<body>
    <applet code=PrimerApplet.class width=300 height=80>
        Applets no soportados.
    </applet>
</body>
</html>
```

PANTALLA 6: Etiqueta APPLET.

La sintaxis de las etiquetas **APPLET** y **PARAM** que situamos en una página HTML que va a incluir un applet es:

```
<APPLET CODE=
        WIDTH= HEIGHT=
        [CODEBASE= ] [ALT= ] [NAME= ]
        [ALIGN= ] [VSPACE= ] [HSPACE= ] >
    <PARAM NAME= VALUE= >
</APPLET>
```

Los atributos que acompañan a la etiqueta **<applet>**, algunos son obligatorios y otros son opcionales. Todos los atributos, siguiendo la sintaxis de html, se especifican de la forma: **atributo=valor**.

Atributos obligatorios

CODE

Indica el fichero de clase que es el applet a ejecutar, que tiene la extensión .class. No se permite un URL absoluto, aunque sí puede ser relativo al atributo opcional **CODEBASE**.

WIDTH

Indica la anchura inicial que el navegador debe reservar para el applet en pixels.

HEIGHT

Indica la altura inicial en pixels. Un applet que disponga de una geometría fija no se verá redimensionado por estos atributos. Por ello, si los atributos definen una zona menor que la que el applet utiliza, únicamente se verá parte del mismo, como si se visualiza a través de una ventana, eso sí, sin ningún tipo de desplazamiento.

Atributos opcionales

CODEBASE

Se emplea para establecer la URL base del applet. En caso de no especificarse, se utilizará el mismo que tiene el documento.

ALT

Funciona exactamente igual que el ALT de la marca , es decir, muestra un texto alternativo, en este caso al applet, en navegadores en modo texto o que entiendan la etiqueta APPLET pero no implementen una máquina virtual Java.

NAME

Otorga un nombre simbólico a esta instancia del applet en la página que puede ser empleado por otros applets de la misma página para localizarlo. Así, un applet puede ser cargado varias veces en la misma página tomando un nombre simbólico distinto en cada momento.

ALIGN

Se emplea para alinear el applet permitiendo al texto fluir a su alrededor. Puede tomar los siguientes valores: LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM y ABSBOTTOM.

VSPACE

Indica el espaciado vertical entre el applet y el texto, en pixels. Sólo funciona cuando se ha indicado **ALIGN = LEFT** o **RIGHT**.

HSPACE

Funciona igual que el anterior pero indicando espaciamiento horizontal, en pixels. Sólo funciona cuando se ha indicado **ALIGN = LEFT** o **RIGHT**.

MASINFO:

Entre las etiquetas **<APPLET></APPLET>** además de etiquetas **<PARAM>** puede ir texto **HTML** que será interpretado por los navegadores que no entienden la marca **APPLET** en sustitución del applet mismo.

PANTALLA 7: ETIQUETA PARAM

Si deseamos pasarle a un applet parámetros desde la página web utilizamos la etiqueta **PARAM**. Los atributos de la etiqueta son:

NAME

Indicamos el nombre del parámetro

VALUE

El valor que queremos darle al parámetro.

Podemos poner tanto parámetros como deseemos.

Estos parámetros tendrán que ser recogidos en el applet. Para recoger un parámetro en un applet utilizamos el método **getParameter(String nombreParámetro)**; Este parámetro devuelve un String con el valor del parámetro. Si queremos convertir la cadena al tipo que deseemos tendremos que utilizar los métodos apropiados.

MASINFO:

En el método **getParameter()** el único argumento que necesita es el nombre del parámetro cuyo valor queremos recuperar. Todos los parámetros se pasan como Strings, en caso de necesitar pasarle al applet un valor entero, se ha de pasar como String, recuperarlo como tal y luego convertirlo al tipo que deseemos.

Tanto el argumento de **NAME** como el de **VALUE** deben ir colocados entre dobles comillas (") ya que son String.

EJEMPLO

Vamos a ver un ejemplo en el que se hace uso de los parámetros. El applet tomará un parámetro y lo utilizará para mostrar un mensaje de saludo. Lo que haremos será pasarle al applet el nombre de la persona a quien queremos saludar. Generamos el código para ello y lo guardamos en el fichero HolaTal.java.

```
import java.awt.Graphics;
import java.applet.Applet;

public class HolaTal extends Applet
{
    String nombre;

    public void init(){
        nombre = getParameter( "Nombre" );
    }

    public void paint( Graphics g ){
        g.drawString( "¡¡ Hola " + nombre + " !!",25,25 );
    }
}
```

Si compilamos el ejemplo obtendremos el fichero **HolaTal.class** que incluiremos en nuestra página Web. Vamos a generar el fichero **HolaTal.html**, en el que incluiremos nuestro applet, y que debería tener el siguiente contenido:

```
<HTML>
  <APPLET CODE=HolaTal.class WIDTH=300 HEIGHT=100>
    <PARAM NAME="Nombre" VALUE="Antonio">
  </APPLET>
</HTML>
```

PANTALLA 8: Métodos

Vamos a ver los métodos más importantes e interesantes que necesitamos para poder realizar applets.

init()

Esta función miembro es llamada al crearse el applet. Es llamada sólo una vez. La clase Applet no hace nada en init(). Las clases derivadas (nuestros applets) deben sobrecargar este método para cambiar el tamaño durante su inicialización, y cualquier otra inicialización de los datos que solamente deba realizarse una vez, cargar imágenes, etc. Deberían realizarse al menos las siguientes acciones:

- ? Carga de imágenes y sonido
- ? El resize del applet para que tenga su tamaño correcto
- ? Asignación de valores a las variables globales

start()

Llamada para activar el applet. Esta función miembro es llamada cuando se visita el applet. La clase Applet no hace nada en este método. Las clases derivadas deberían sobrecargarlo para comenzar una animación, sonido, lanzar a ejecución threads, etc.

stop()

Llamada para detener el applet. Se llama cuando el applet desaparece de la pantalla. La clase Applet no hace nada en este método. Las clases derivadas deberían sobrecargarlo para detener la animación, el sonido, parar la ejecución de los threads lanzados en start, etc.

destroy()

Esta función miembro es llamada cuando el applet no se va a usar más. La clase Applet no hace nada en este método. Las clases derivadas deberían sobrecargarlo para hacer una limpieza final. Los applet multithread deberán usar destroy() para "matar" cualquier thread del applet que quedase activo.

getParameter(String attr)

Este método carga los valores pasados al applet vía la marca PARAM de HTML. El argumento String es el nombre del parámetro que se quiere obtener. Devuelve el valor que se le haya asignado al parámetro; en caso de que no se le haya asignado ninguno, devolverá **null**. Una vez que se ha capturado el parámetro, se utilizan métodos de cadena o de números para convertir el valor obtenido al tipo adecuado.

getDocumentBase()

Indica la ruta http, o el directorio del disco, de donde se ha recogido la página HTML que contiene el applet, es decir, el lugar donde está la hoja en todo Internet o en el disco.

getCodeBase()

Indica la ruta http, o el directorio del disco, de donde se ha cargado el código bytecode que forma el applet, es decir, el lugar donde está el fichero .class en todo Internet o en el disco.

getAppletContext()

Este método devuelve un objeto, construido por el navegador, que implementa la interfaz **AppletContext**. Esta interfaz tiene los siguientes métodos, entre otros:

- ? **Applet getApplet(String)**. Devuelve el objeto applet con el nombre especificado en el parámetro. Si el applet no existe en la página devuelve null.
- ? **Enumeration getApplets()**. Devuelve un iterator con todos los applets en ejecución.
- ? **AudioClip getAudioClip(URL)**. Obtiene un fichero de sonido.
- ? **Image getImage(URL)**. Obtiene una imagen.

- ? **void showDocument(URL,target)**. Indica al navegador que visualice la página especificada.
- ? **void showStatus(String)**. Muestra la cadena recibida como parámetro en la barra del navegador de estado del navegador.

EJEMPLO

```
public void init()
{
    String pv;
    pv = getParameter( "velocidad" );
    if( pv == null )
        velocidad = 10;
    else
        velocidad = Integer.parseInt( pv );
}
```

PANTALLA: RESUMEN

A lo largo de esta unidad habrás aprendido que:

- ? Un applet es una mínima **aplicación** Java diseñada para ejecutarse en un **navegador** web.
- ? Todo **applet** tiene que ser una clase que herede de la clase **java.applet.Applet**.
- ? Según el navegador web que estemos usando los applets tendrán una serie de **restricciones**, como por ejemplo no poder leer ciertas propiedades del sistema.
- ? Los applets además tienen una serie de **capacidades** que otras aplicaciones no tienen como por ejemplo hacer conexiones al host del que fueron cargados.
- ? Para que el lenguaje HTML pueda soportar los applets de Java, tenemos a nuestra disposición las etiquetas `<APPLET>` y `<PARAM>`.
- ? Para poder crear un applet tenemos a nuestra disposición una serie de métodos que debemos de utilizar, incluso para el applet más sencillo. Entre estos métodos nos encontramos con los métodos `INIT()`, `Start()`, `Stop()`. y `destroy()`

Unidad Didáctica: JavaScript

Pantalla: 1. Objetivos formativos

- ? Conocer las posibilidades que el lenguaje javascript aporta a las páginas HTML.
- ? Saber programar scripts dentro de una página HTML para desempeñar operaciones básicas.
- ? Saber emplear las distintas estructuras de control de ejecución de un programa.
- ? Saber programar funciones básicas y hacer que se ejecuten a través de acciones provocadas por el usuario.

Pantalla: 2. JavaScript: Características principales

Definición:

Es un lenguaje de programación pensado y creado únicamente para su uso en Internet. Se inserta dentro del código HTML de las páginas web, y es interpretado y ejecutado por el navegador del usuario. Nos permite crear efectos especiales en las páginas y definir interactividades con el usuario.

Desarrollo:

Sus características más importantes son:

- ? **JavaScript es un lenguaje interpretado**, no es compilado, es leído e interpretado directamente por el navegador como código fuente cuando éste lee la página.
- ? **JavaScript es un lenguaje de guiones**. Un programa con pequeños guiones (scripts), estos miniprogramas se interpretan línea por línea mientras que la aplicación sigue su ejecución normal.
- ? **JavaScript es un lenguaje orientado a eventos**. Este lenguaje está capacitado para detectar un sin número de eventos y reaccionar a ellos. Al ser guiado por eventos, no tenemos una función principal que se ejecute por delante de las demás, sino que tendremos funciones.
- ? **JavaScript es un lenguaje orientado a objetos**. Aunque sería mejor decir que su estructura está basada en objetos. Al margen de cuestiones técnicas el caso es que tiene una biblioteca de objetos predefinidos.

Masinfo:

Javascript **no tiene nada que ver con Java**. Son productos totalmente distintos y no guardan entre sí más relación que la sintaxis idéntica y poco más. Java es mucho más complejo y potente, es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con Javascript sólo podemos escribir programas para que se ejecuten en páginas web.

Pantalla: 3. Introduciendo Scripts

Introducción:

La programación de Javascript se realiza dentro del propio código HTML, dentro de las etiquetas **<SCRIPT>... </SCRIPT>**. Todo el código Javascript de una página Web ha de ser introducido entre estas dos etiquetas.

Desarrollo:

El atributo que aplicaremos a la etiqueta **<SCRIPT>** es **language**. Si no lo indicamos, el navegador entenderá que el lenguaje con el que se está programando es Javascript. La sintaxis para introducir código JavaScript en una página HTML sería:

```
<SCRIPT language="JavaScript">  
    Código JavaScript  
</SCRIPT>
```

Cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va ejecutando una después de otra.

Ejemplo:

```
<HTML >  
  <HEAD>  
    <TITLE>JavaScript 1</TITLE >  
    <SCRIPT language="javascript">  
      alert("Hola mundo en Javascript");  
    </SCRIPT >  
  </HEAD>  
  <BODY >  
    Esta es mi primera página con un script  
  </BODY >  
</HTML >
```


Pantalla: 4. Declaración y uso de variables

Definición:

El concepto de variable debe verse como un espacio **en memoria donde se almacena un dato**, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas. Las variables pueden tener cualquier nombre, con cualquier sucesión de letras, números o palabras

Desarrollo:

Para declarar una variable se utiliza la palabra **var** seguida del nombre que se le da a la variable, a partir de ese momento el programa sabe que esa variable existe y puede ser utilizada asignándole un valor.

Aunque es aconsejable hacerlo para mayor claridad, en javascript no existe esa obligación, y así también se puede asignar un valor a la variable cuando se está declarando, o simplemente asignarle un valor sin haberla declarado previamente con la palabra **var**. Todos los ejemplos siguientes son válidos:

```
var operando1 // declaramos la variable operando1  
operando1=32 // le asignamos un valor. Puede haber sido declarada  
previamente o no.  
var operando1= 32 // declaramos la variable y le asignamos a la vez un valor
```

Masinfo:

Atención especial debes prestar a las mayúsculas y minúsculas, pues es capaz de distinguirlas. Todas las palabras que usemos: variables, funciones, palabras reservadas, etc..., tendrán que ser tecleadas correctamente para poder ser adecuadamente interpretadas. Esta propiedad es fuente de múltiples errores.

Pantalla: 5. Tipos de datos

Definición:

En una variable podemos introducir varios tipos de información: **texto, números enteros o reales**, etc. A estas distintas clases de información se les conoce como **tipos de datos**. JavaScript tiene la peculiaridad de ser un lenguaje débilmente tipado, esto quiere decir que las variables no están forzadas a guardar un tipo de datos en concreto. Podemos introducir cualquier tipo de información en una variable, e ir cambiando de un tipo a otro sin ningún problema.

Desarrollo:

A continuación vemos los **tipos de datos** que JavaScript reconoce:

- ? **Tipo de datos numérico.** En la mayoría de los lenguajes de programación, a la hora de definir una variable numérica, debemos elegir entre varios tipos según sea un valor entero, flotante, corto, largo, con signo, sin signo, etc. En JavaScript no es necesario. Todos los números son del tipo numérico, independientemente de la precisión que tengan.
- ? **Tipo de datos booleano.** Este tipo de datos sirve para guardar un si o un no o dicho de otro modo, un verdadero (**true**) o un falso (**false**). Se utiliza para realizar operaciones lógicas. De nuevo en este punto tendremos que tener precaución con el uso de las mayúsculas y minúsculas. JavaScript reconoce únicamente **true** y **false**, en minúsculas.
- ? **Tipo de datos cadena de caracteres.** Otro tipo de datos que puede tener una variable es el que sirve para guardar un texto. Los textos se escriben siempre entre comillas dobles("") o simples (''). Todo lo que se coloque entre comillas es tratado como una cadena de caracteres, aunque sean números, al guardarlos como texto no podremos hacer cálculos matemáticos con ellos, solo las variables numéricas sirven para hacer cálculos matemáticos.
- ? **El valor nulo (null)** es el valor que tendrá una variable declarada pero a la que todavía no se le ha asignado ningún valor. Debe distinguirse del término **undefined**, que alude a una variable no definida.

Ejemplo:

```
<HTML>
  <HEAD>
    <TITLE>JavaScript 2</TITLE>
    <SCRIPT language="javascript">
      var cantidad = 1000; //tipo numérico
      var nombre = "Francisco"; // tipo carácter
      var valido = true; // tipo booleano
      var nodefinida; //tipo null

    </SCRIPT>
  </HEAD>
  <BODY>
    En este Script se declaran variables de todos los tipos
  </BODY>
</HTML>
```

Pantalla: 6. Operadores

Definición:

Consisten en conjunto de funciones que sirven para hacer los cálculos y operaciones necesarias con las variables para llevar a cabo los objetivos del programa.

Desarrollo:

Los operadores se pueden clasificar según el tipo de acciones que realizan, así tenemos:

? **Operadores aritméticos**

Operador	Descripción
+	Permite la suma de valores
-	Resta de valores, también indica un número negativo si se usa con un sólo valor: -23
*	Multiplicación de valores
/	División de dos valores
%	Módulo: El resto de la división de dos números
--	Decremento en una unidad, utilizado con un solo operando
++	Incremento en una unidad, se utiliza con un solo operando

Comentario especial merecen los dos últimos operadores aritméticos. Su única función consiste en aumentar o disminuir el valor de una variable numérica en una unidad.

? Operadores de asignación

Operador	Descripción	Ejemplo
=	Asigna la parte de la derecha a la de la izquierda	precio = 100
+=	Suma la izquierda y derecha y asigna el resultado a la parte de la izquierda	precio += 200 //incrementa precio en 200, ahora vale 300
-=	Resta la izquierda y derecha y asigna el resultado a la parte de la izquierda	precio -= 200 //decrementa precio en 200, ahora vale 100
*=	Multiplica la izquierda por la derecha y asigna el resultado a la parte de la izquierda	precio *= 2 //multiplica precio por 2, ahora vale 200
/=	Divide la izquierda por la derecha y asigna el resultado a la parte de la izquierda	precio /= 4 //Divide precio por 4, ahora vale 50
%=	Divide la izquierda por la derecha y asigna el valor del resto de la división a la parte de la izquierda	precio %= 10 //Divide precio por 10, ahora vale 0

? Operadores de cadenas

+ Concatena dos cadenas, pega la segunda cadena a continuación de la primera.

```
cadena1 = "hola"  
cadena2 = "mundo"  
cadenaConcatenada = cadena1 + cadena2 //cadena concatenada vale "holamundo"
```

Un detalle importante que se puede ver en este caso es que el operador + sirve para dos usos distintos, si sus operandos son números los suma, pero si se trata de cadenas las concatena. Si por error usamos el operador + con una variable de texto y otra numérica javascript asume que se desea realizar una concatenación y trata a los dos operandos como si de cadenas de caracteres se trataran.

? Operadores lógicos

Los operadores lógicos son los operadores del álgebra de Bull o booleanos. sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso para realizar este tipo de operaciones se utilizan operandos booleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false).

Operador	Descripción
&&	AND (Y) lógico: devuelve verdadero (true) si los dos operandos son verdaderos, y falso (false) en caso contrario
	OR (O) lógico devuelve verdadero si uno de los dos operandos o ambos son verdaderos, y falso en caso contrario
!	NOT (No) lógico devuelve verdadero si el operando es falso, y falso si es verdadero

? Operadores de comparación

Los operadores de comparación devuelven siempre verdadero (**true**) o falso (**false**). Estas expresiones se utilizan para realizar una acción u otra en función de la comparación de varios elementos, por ejemplo si un numero es mayor que otro o si son iguales. Son muy utilizados en las expresiones condicionales para toma de decisiones:

Operador	Descripción
==	Comprueba si dos operadores son iguales y si es así devuelve true. Un error bastante común es confundirlo con el operador de asignación (=)
!=	Comprueba si dos operadores son distintos y si es así devuelve true.
>	devuelve true si el primer operador es mayor que el segundo
<	Devuelve true cuando el primer operador es menor que el segundo
>=	Devuelve true si el primer operador es mayor o igual que el segundo
<=	Devuelve true si el primer operador es menor o igual que el segundo

? Operadores a nivel de bit

Estos son poco corrientes. Se usan a nivel binario, es decir , para efectuar operaciones con ceros y unos.

Operador	Descripción
&	AND, Y de bits
^	Xor de bits
	OR, O de bits
~	NOT de bits
<<	Rotación izquierda de bits
>>	Rotación derecha

Ejemplo:

```
<HTML>
  <HEAD>
    <TITLE>JavaScript 2</TITLE>
    <SCRIPT language="javascript">
      //Declaración de variables
      var cantidad1 = 5;
      var cantidad2 = 3;
      var resultado = 0;
      //Operaciones aritméticas
      resultado = cantidad1 + cantidad2;
      alert(resultado);
      resultado = cantidad1 - cantidad2;
      alert(resultado);
      resultado = cantidad1 * cantidad2;
      alert(resultado);
      resultado = cantidad1 / cantidad2;
      alert(resultado);

    </SCRIPT>
  </HEAD>
  <BODY>
    En este script se realizan una serie de operaciones aritméticas a
    modo de ejemplo. La función alert() permite ver el resultado de cada operación
  </BODY>
</HTML>
```

Pantalla: 7. Estructuras de control (I)

Definición:

En los programas generalmente necesitaremos hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir la misma línea de código una y otra vez. Para ello se utilizan las **estructuras de control**.

Desarrollo:

Expresiones **condicionales**:

- ? **Estructura if.** Esta estructura permite la evaluación de una expresión. Si la evaluación da resultado positivo se realizan las acciones relacionadas con el caso positivo. Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de resultados negativos. La sintaxis es la siguiente:

```
if (expresión) {  
    acciones a realizar en caso positivo  
} else {  
    acciones a realizar en caso negativo  
}
```

La expresión o condición a evaluar se coloca siempre entre paréntesis y está compuesta por variables que se combinan entre si mediante operadores de comparación: **if (numero1 == numero2)**

- ? **Estructura switch.** Es otra expresión disponible para tomar decisiones. se utiliza cuando tenemos múltiples posibilidades como resultado de la evaluación de una sentencia. Su sintaxis es la siguiente:

```
switch (expresión) {  
    case valor1:  
        Sentencias a ejecutar si la expresión tiene como  
        valor a valor1;  
        break;  
    case valor2:  
        Sentencias a ejecutar si la expresión tiene como  
        valor a valor2;  
        break;  
    case valorN:  
        Sentencias a ejecutar si la expresión tiene como  
        valor a valorN;  
        break;  
    default:  
        Sentencias a ejecutar si el valor no es ninguno de  
        los anteriores;  
}
```

Ejemplo:

```
<HTML>
  <HEAD>
    <TITLE> Condicionales </TITLE>
    <SCRIPT language="javascript">
      //Declaración de variables
      var cantidad1 = 5;
      var cantidad2 = 3;
      var resultado = 0;

      //Estructuras condicionales
      if (cantidad1 > cantidad2) {
        alert("la cantidad 1 es mayor que la cantidad 2");
      }
      else {
        alert("la cantidad 1 es menor que la cantidad 2");
      }

      switch (resultado) {
        case 5:
          alert("se ha introducir la cantidad 1");
          break;
        case 3:
          alert("se ha introducir la cantidad 2");
          break;
        default:
          alert("no se introdujo ninguna cantidad");
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    En este ejemplo se utilizan las condicionales if y switch
  </BODY>
</HTML>
```


Pantalla: 8. Estructuras de control (II)

Definición:

Los bucles se utilizan para realizar ciertas acciones repetidamente hasta que se cumple una condición.

Desarrollo:

En javascript existen varios tipos de bucles, cada uno está indicado para un tipo de iteración distinto y son los siguientes:

- ? **Repeticiones con for.** Se utiliza cuando sabemos seguro el número de veces que queremos que se ejecute la sentencia. La sintaxis del bucle se muestra a continuación:

```
for (valor inicial; condición; actualización){  
  instrucciones a ejecutar en el bucle  
}
```

Todos los bucles for tienen tres elementos principales, incluidos entre paréntesis y que son:

- ? El **valor inicial** es una variable que señala la primera iteración del bucle.
- ? La **condición** es una expresión que se evalúa cada vez que se ejecuta una repetición del bucle y detiene el mismo en el momento de ser falsa.
- ? La **actualización** es la expresión con la que modificamos el valor inicial en cada repetición. Por lo general, consiste en incrementar ó decrementar la variable.

Un ejemplo real de estos elementos sería: **for (i=0;i<=10;i++)**. Se inicia el bucle con la variable i valiendo 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Estos bucles también pueden anidarse para hacer procesamientos un poco más complejos.

- ? **Bucle while.** Es la estructura más sencilla para crear repeticiones. se utiliza cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo que el bucle **for**, pues sólo se indica la condición que se tiene que cumplir para que se realice una iteración. **while** es un bucle que puede que no se realice ninguna vez, pues la condición se evalúa el principio del bucle.

```
while (condición){  
  sentencias a ejecutar  
}
```

- ? **Bucle do...while.** Se utiliza cuando no sabemos cuantas veces se habrá de ejecutar el bucle, igual que el bucle **while**, con la diferencia de que el bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución La sintaxis es la siguiente.

```
do {  
  sentencias del bucle  
} while (condición)
```

Ejemplo:

```
<HTML>
  <HEAD>
    <TITLE>Bucle</TITLE>
    <SCRIPT language="javascript">
      //Declaración de variables
      var cantidad1 = 5;
      var cantidad2 = 3;
      var resultado = 0;

      //Bucles
      for(x=1;x<11;x++) {
        alert("Esta operación se a realizado" + x +
"vez/veces");
      }

      while (x<5) {
        x++;
        alert("Esta operación se a realizado" + x +
"vez/veces");
      }

      do {
        x++;
        alert("Esta operación se a realizado" + x +
"vez/veces");
      } while (x<5)

    </SCRIPT>
  </HEAD>
  <BODY>
    En este ejemplo se utilizan los bucles for y while y do while
  </BODY>
</HTML>
```

Pantalla: 9. Funciones

Definición:

Una función es una serie de instrucciones que englobamos dentro de un mismo proceso, creado con la finalidad de realizar una determinada acción.

Desarrollo:

Para definir una función se escribe la palabra **function** seguida por el nombre de la función y unos paréntesis; al igual que en las variables el nombre es sensible a mayúsculas y minúsculas, y puede tener números, letras e incluir el carácter `_`. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones son obligatorias, además es útil colocarlas siempre como se verá en el ejemplo, para que se vea fácilmente la estructura de instrucciones que engloba la función.

Para ejecutar la función la tenemos que llamar utilizando su nombre seguido de los paréntesis:

```
function suma(valor1,valor2){  
  var resultado  
  resultado = valor1 + valor2  
  return resultado  
}
```

Pantalla: 10. Eventos

Definición:

Un evento es una **acción que realiza el usuario** (hacer click, posicionarse con el ratón sobre un lugar determinado, enviar un formulario,...). ante la cual puede realizarse algún proceso. Estos eventos se capturan con los manejadores de eventos que son mecanismos mediante los que se detectan las acciones y llaman automáticamente a la función que haya asociada para que se realice.

Desarrollo:

Los eventos se sitúan dentro del código HTML de nuestra página y "llaman" a nuestras funciones Javascript:

`Llamada a función`

Dentro de un hiperenlace estaremos llamando a la función mifuncion() al hacer clic sobre él.

En la siguiente tabla puedes ver los manejadores de eventos más utilizados, la causa que provoca el evento, las etiquetas HTML en las que puede insertarse el manejador del evento y los objetos predefinidos de Javascript que admiten el evento:

Evento	Causa del evento	Directivas asociadas	Objetos que lo admiten
onLoad	Se carga una página (entrar)	<BODY>, 	Image, window
onUnload	Se descarga una página (salir)	<BODY>	window
onMouseOver	El puntero pasa sobre algo	<A HREF>, <AREA>	Link
onMouseOut	El puntero sale de algo	<A HREF>, <AREA>	Link
onSubmit	Se envía un formulario	<FORM>	Form
onClick	Se pulsa el ratón sobre algo	<INPUT TYPE="...">, <AREA>, <A HREF>	Button, document, Checkbox, Link, Radio, Reset, Submit

Ejemplo:

```
<HTML>
  <HEAD>
    <TITLE>Funciones y eventos</TITLE>
    <SCRIPT language="javascript">
      function holamundo() {
        alert("Hola mundo");
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <a href="#" onClick="javascript:holamundo();">Llamada a la
función hola mundo() </a>
  </BODY>
</HTML>
```

Pantalla: 11. Resumen

A lo largo de esta unidad habrás aprendido:

- ? Javascript es un lenguaje que sólo comparte con Java la sintaxis y que nos **permite introducir pequeños programas** (scripts) dentro de nuestras páginas Web.
- ? Las **variables** que podemos utilizamos en Javascript pueden ser:
 - o Numéricas (cifras)
 - o Cadenas de caracteres (texto)
 - o Booleanas (positivo o negativo)
 - o Nulas (variables no definidas)

Su declaración **no es obligatoria** pero si muy conveniente de cara a la claridad de nuestro código.

- ? Los **operadores** son funciones que nos permiten llevar a cabo operaciones matemáticas y de comparación. Existen los siguientes tipos de operadores:
 - o Aritméticos
 - o Asignación
 - o Cadena
 - o Lógicos
 - o Comparación
 - o A nivel de bit
- ? Las **estructuras de control** son los mecanismos que nos permiten conducir la ejecución de nuestros programas según un propósito. Existen dos estructuras de control fundamentales:
 - o **Condicionales**. La ejecución de un script depende del cumplimiento de una determinada condición se cumpla o no.
 - o **Bucles**. Permiten repetir una determinada operación tantas veces como deseemos.
- ? Las **funciones** son un conjunto de sentencias **encapsuladas** cuya ejecución está provocada por una acción del visitante de nuestra página, estas acciones reciben el nombre de **eventos**.