

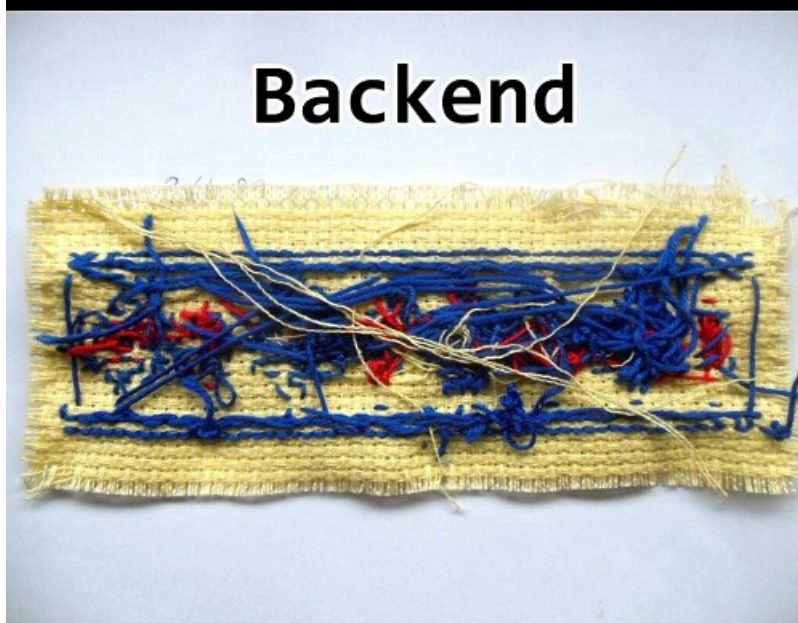
DESARROLLO EN ENTORNO CLIENTE/SERVIDOR

UD2 – FUNDAMENTOS DE PROGRAMACIÓN WEB

Frontend



Backend



UD2 - Fundamentos de programación web

1. LENGUAJES EMBEBIDOS EN HTML.....	3
1.1. Tecnologías para páginas dinámicas.....	4
1.1.1. Tecnologías de Servidor.....	4
1.1.2. Tecnologías de Cliente.....	7
1.1.3. Tecnologías que vamos a ver en este curso.....	8
2. INSERCIÓN DE CÓDIGO EN HTML Y CREACIÓN DE PÁGINAS DINÁMICAS.....	9
2.1. Configuración del entorno.....	9
2.1.1. Edición de código.....	9
2.1.2. Ejecución del código en JavaScript.....	9
2.1.3. Ejecución del código en PHP y Python.....	10
2.2. Insertando código en JavaScript.....	16
2.3. Insertando código en PHP.....	18
2.4. Insertando código en Python.....	20
3. COMENTARIOS EN EL CÓDIGO.....	23
3.1. Comentarios en JavaScript.....	24
3.2. Comentarios en PHP.....	25
3.3. Comentarios en Python.....	25
4. VARIABLES.....	26
4.1. Variables en JavaScript.....	26
4.2. Variables en PHP.....	28
4.3. Variables en Python.....	30

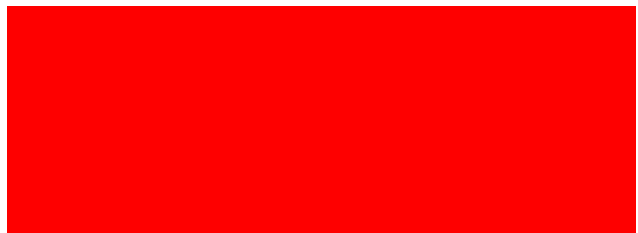
1. LENGUAJES EMBEBIDOS EN HTML.

En primero ya se vio como generar una página web en HTML. Si tenemos un archivo, en el que escribimos con la sintaxis correcta, etiquetas HTML y lo abrimos con un navegador, veremos que en vez de mantener el contenido escrito, el navegador lo interpreta y nos muestra no el texto escrito, si no la interpretación del mismo. Por ejemplo:

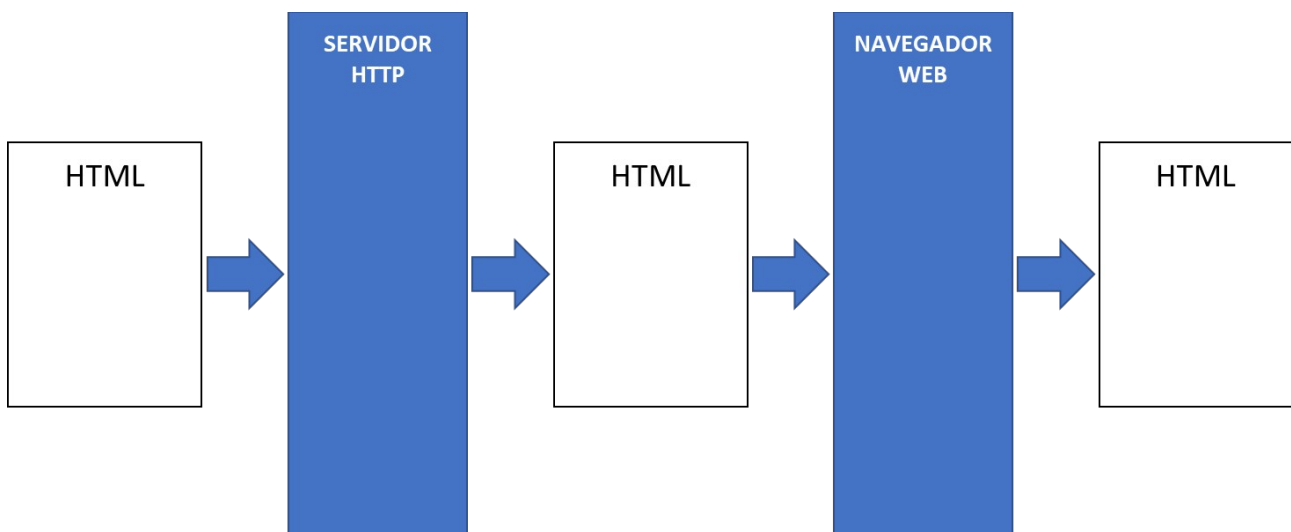
Si escribimos el siguiente código:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 </head>
5 <body style="background-color: #FF0000;">
6 </body>
7 </html>
```

El navegador mostrará esto:



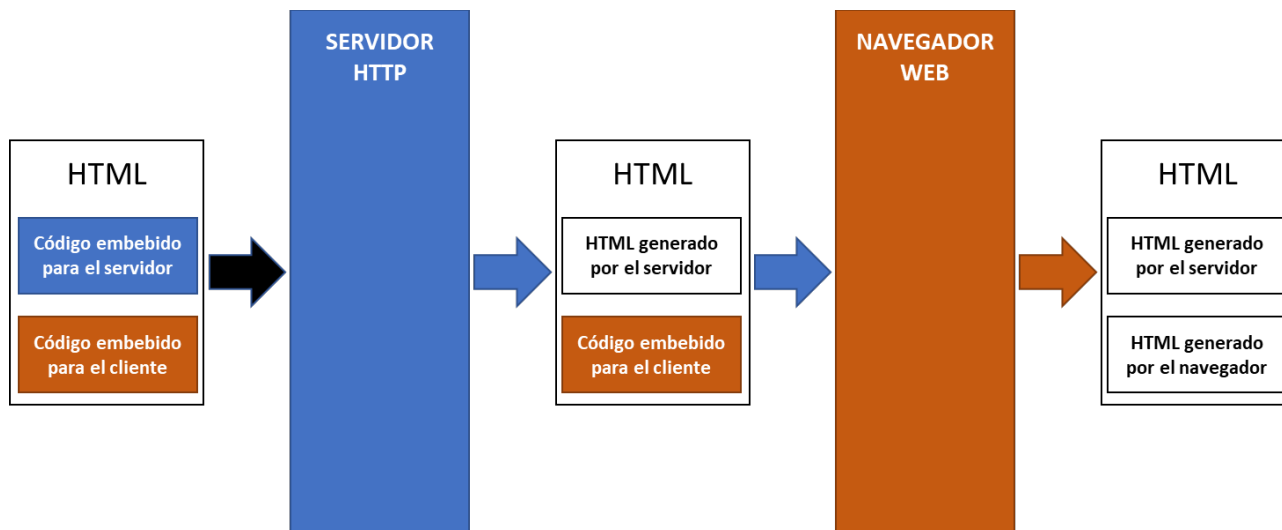
Pero por muchas veces que solicitemos la página al servidor, obtenemos siempre lo mismo:



La página HTML la lee el servidor, y tal cuál se la envía al navegador que, una vez interpretada como HTML, la interpreta y la muestra al cliente. No se ha producido ninguna transformación por el camino. Como se vio en el tema anterior, a esto se le denomina **página estática**.

Pero existe otra posibilidad, y es que insertemos código embebido dentro del HTML. Se define código embebido como partes o fragmento de código fuente de uno o varios lenguajes de programación diferente al lenguaje HTML, pero que se incluye dentro de la página HTML. De esta

forma, se incluye código que el servidor o el navegador interpretará, según sea el destinatario de ejecutarlo, y lo sustituirá por códigos HTML que el navegador interpretará y mostrará al usuario:



En este segundo caso, aunque la página almacenada en el servidor sea siempre la misma, cuando el servidor la lee, interpreta el trozo de código que le corresponde y modifica esa página de forma específica. Igualmente ocurre con el navegador web. El resultado es una página que mostrará un resultado que puede variar o adaptarse. A esto lo denominamos **página dinámica**.

1.1. Tecnologías para páginas dinámicas

Existen varias tecnologías que se han usado para crear páginas dinámicas.

1.1.1. Tecnologías de Servidor

CGI (Common Gateway Interface)

Es la primera implementación que se realizó, y hoy está algo en desuso, principalmente por cuestiones de eficiencia (lanza un proceso independiente, lo cuál es costoso en tiempo de ejecución, consumo de memoria, etc.), pero los servidores siguen manteniendo esta posibilidad y, en algún caso, podría ser la solución ideal.

Cuando se invoca cierta URL del servidor, en realidad se le está pidiendo que ejecute un programa situado en el servidor y se le pasan los datos recibidos por el servidor. Ese programa puede estar programado en cualquier lenguaje que acepte el sistema operativo del servidor. El

programa ha de producir una respuesta en HTML en su salida estándar (por ejemplo, en java debería hacerse algo como:

```
1  class holaMundoHTML
2  {
3      // Imprime una página HTML que muestra "Hola Mundo".
4      public static void main(String args[])
5      {
6          //En la variable salida vamos construyendo la página HTML
7          String salida="Content-type: text/html\n\n";
8          salida = salida+"<!DOCTYPE html>";
9          salida = salida+"<html lang='es'>";
10         salida = salida+"<head>";
11         salida = salida+"</head>";
12         salida = salida+"<body>";
13         salida = salida+"HOLA MUNDO";
14         salida = salida+"</body>";
15         salida = salida+"</html>";
16
17         //Cuando tenemos toda la salida preparada,
18         //la devolvemos por la salida estándar
19         System.out.println(salida);
20     }
21 }
```

Nota: Al ser un CGI, se requiere crear la cabecera http con el valor mínimo requerido (en este caso el tipo *MIME* expresado como *text/html* en este caso).

Servlets de Java (e interfaz ISAPI de Microsoft)

Para resolver los problemas de eficiencia, Sun Microsystems (la empresa que desarrolló java) creo los servlets. Un servlet es un programa Java que se ejecuta en un servidor Web y construye o sirve páginas web. De esta forma se pueden construir páginas dinámicas, basadas en diferentes fuentes variables: datos proporcionados por el usuario, fuentes de información variable (páginas de noticias, por ejemplo), o programas que extraigan información de bases de datos.

Comparado con un CGI, un servlet es más sencillo de utilizar, más eficiente (se arranca un hilo por

cada petición y no un proceso entero), más potente y portable. Con los servlets podremos, entre otras cosas, procesar, sincronizar y coordinar múltiples peticiones de clientes, reenviar peticiones a otros servlets o a otros servidores, etc.

Microsoft desarrolló una tecnología parecida, denominada Internet System API (ISAPI). Es una interfaz, de programación de aplicaciones (API) para el servidor web de Microsoft, IIS (Internet Information Server). A partir de la versión 6.0

La ISAPI permite que los programadores puedan desarrollar aplicaciones basadas en web que se procesen mucho más rápidamente que los programas CGI. Esto es así porque están más integrados con el servidor web.

Lenguajes de SCRIPT

La tercera evolución, son los lenguajes script de servidor. Estos son lenguajes desarrollados para incrustarse o embeberse dentro de una página HTML. Si en CGI o servlets el servidor llama a un proceso externo para ejecutar el programa, con los lenguajes de script es el propio servidor web el que lee, analiza y ejecuta esa parte de programación incluida en el script.

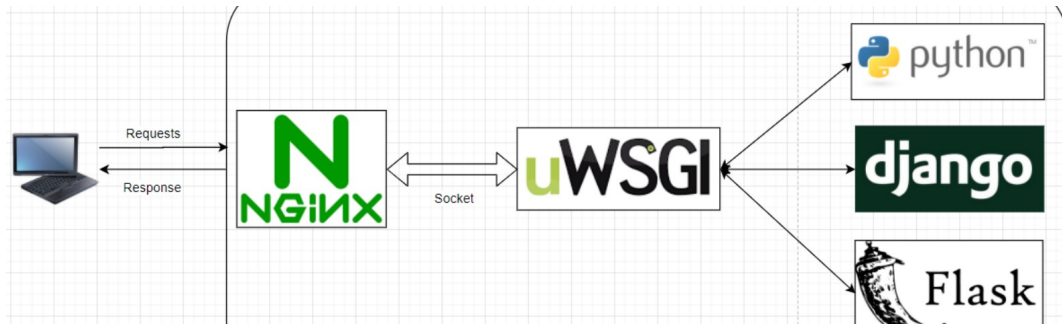
Existen varios lenguajes de script, tales como ASP (Microsoft), Net.Data (IBM) o PHP siendo el más extendido.

Web Server Gateway Interface (WSGI) para Python

WSGI es una especificación común para servidores web el cual permite que los servidores web reenvíen solicitudes a aplicaciones web o frameworks escritos en el lenguaje de programación Python. Se puede ver como una extensión de CGI, pero mucho más eficiente. Usualmente, cuando se instala un framework de programación web en Python, como Flask o Django, también se instala y configura un servidor WSGI. Algunos de los más usados son Gunicorn, uWSGI o `mod_wsgi`.

El cliente hace una petición al servidor web (en el siguiente esquema, a nginx), éste, al detectar

que es una petición de ejecución de Python, se la pasa al servidor WSGI (uWSGI en este caso), que ya invoca a un código en Python directo, o a un framework como pueden ser Django o Flask.



1.1.2. Tecnologías de Cliente

Applets de Java

Fue la primera tecnología de páginas dinámicas en el lado servidor. Un Applet es un programa que puede incrustarse en un documento HTML. Cuando un Navegador carga una página Web que contiene un Applet, éste se descarga en el navegador Web y comienza a ejecutarse, lo cuál nos permite crear programas que cualquier usuario puede ejecutar. Uno de los requisitos para el applet funcione es tener instalado Java y que esté activado a través del explorador web. Es una tecnología en desuso y algunos navegadores (como Chrome) ni siquiera dan soporte a applets.

Active X

Es una tecnología parecida a los applets de java, pero desarrollada por Microsoft, funcionando exclusivamente en Windows. La principal diferencia es que un applet, por protección del navegador, no podía acceder a ningún recurso del cliente, mientras que un componente activex puede acceder a cualquier recurso del ordenador cliente, haciendo de los mismos un potencial agujero de seguridad. Hoy día también están casi completamente en desuso.

Adobe Flash

Adobe Flash (inicialmente FutureSplash Animator, después Macromedia Flash5 hasta 2005) fue

plataforma de software usada para la producción de animaciones, aplicaciones de internet enriquecidas, aplicaciones de escritorio, aplicaciones móviles, juegos para móviles, y reproductores de video web embebidos. Flash muestra textos, gráficos vectoriales e imágenes de mapa de bits para crear animaciones, videojuegos y aplicaciones. Permite la transmisión de audio y video, y permite capturar las entradas del mouse, teclado, micrófono y cámara.

Actualmente está totalmente obsoleta y los navegadores no tienen soporte de Flash, tanto por cuestiones de rendimiento como de seguridad.

JavaScript

Es con diferencia el lenguaje de script de cliente más usado y en constante desarrollo. JavaScript es el lenguaje de programación encargado de dotar de mayor interactividad y dinamismo a las páginas web. Cuando JavaScript se ejecuta en el navegador, no necesita de un compilador. El navegador lee directamente el código, sin necesidad de terceros. Por tanto, se le reconoce como uno de los tres lenguajes nativos de la web junto a HTML (contenido y su estructura) y a CSS (diseño del contenido y su estructura).

No conviene confundir JavaScript con Java, que es un lenguaje de programación muy diferente. La confusión proviene del nombre, registrado por la misma empresa creadora de Java (Sun Microsystems). JavaScript (JS) se creó posteriormente, y la empresa norteamericana lo que hizo simplemente fue cambiar el nombre que le habían puesto sus creadores al comprar el proyecto (LiveScript). El lenguaje de programación Java está orientado a muchas más cosas que la web desde sus inicios.

1.1.3. Tecnologías que vamos a ver en este curso

En este curso nos vamos a centrar en PHP y Python, para la parte servidor, y JavaScript, para la parte cliente.

2. INSERCIÓN DE CÓDIGO EN HTML Y CREACIÓN DE PÁGINAS DINÁMICAS.

Para que se ejecute el código que se pone en el HTML (tanto en la parte servidor como en la parte cliente) el elemento encargado de ejecutarlo ha de saber qué lo que se le solicita no ha de enviarlo/mostrarlo sin más, si no que se requiere un proceso de ejecución previo.

Esto va a variar según estemos en el lado del servidor o en el lado del cliente.

2.1. Configuración del entorno

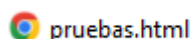
Antes de poder ejecutar código, necesitamos una mínima configuración del entorno de trabajo. Siendo relativamente sencilla en el caso del lado cliente (JavaScript) y un poco más compleja en el lado servidor (PHP y Python).

2.1.1. Edición de código

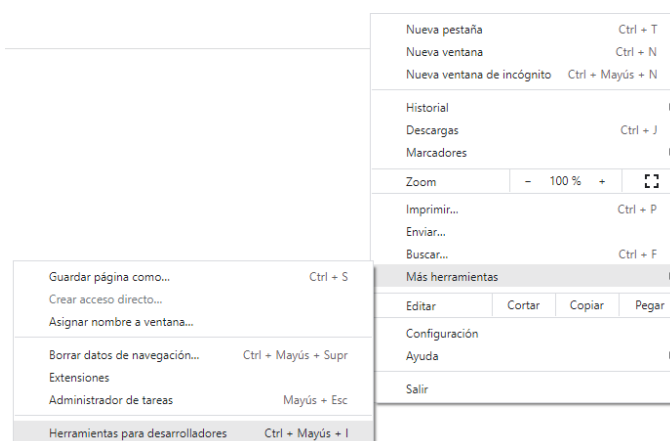
La edición del código la haremos a través de un editor de código fuente. Algunos de los editores más usados son Eclipse, NetBeans, Notepad++ o SublimeText. Aunque para los ejemplos que iremos mostrando, el entorno que se usará es Visual Studio Code, que aunque sea desarrollado por Microsoft, es código abierto, multiplataforma y tiene múltiples extensiones para adaptarse a la programación en diferentes lenguajes.

2.1.2. Ejecución del código en JavaScript

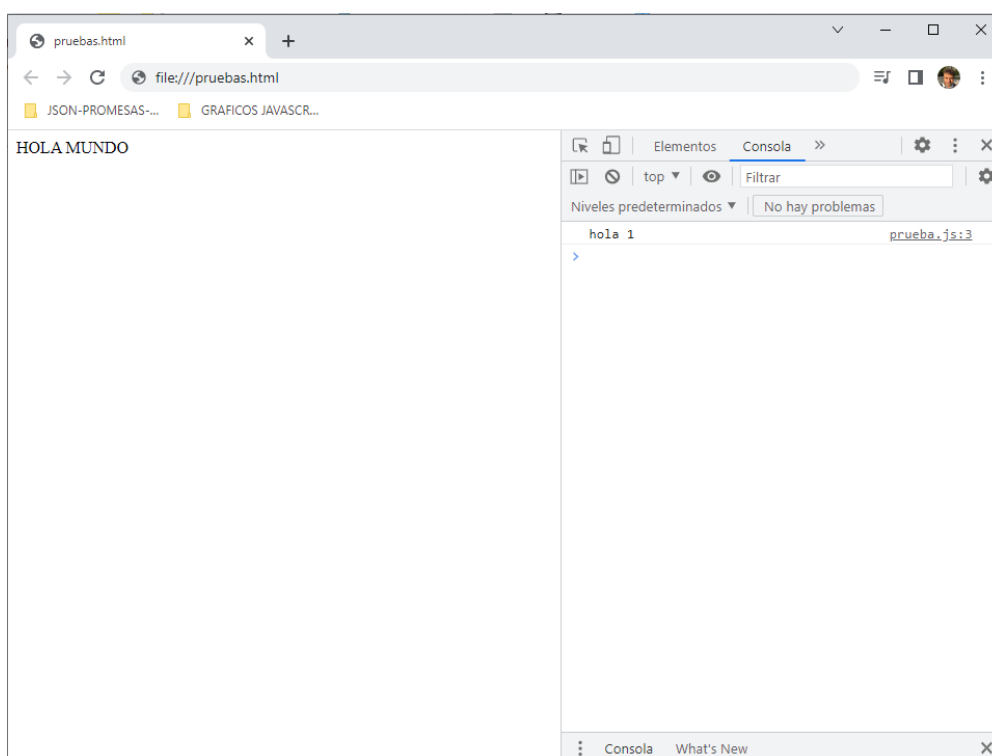
Simplemente necesitaremos un navegador web (Chrome o Firefox van a ser las opciones más usuales). Siempre procederemos a cargar una página HTML que incluirá el código JavaScript o una llamada al fichero que lo contenga.



Por ejemplo, dado el fichero anterior, simplemente se ha de hacer doble click sobre el mismo, y el navegador mostrará el resultado. Como vamos a trabajar con programas en JavaScript es recomendable abrir la consola del navegador, que se encuentra en las opciones del navegador y ahí "Más herramientas/Herramientas para desarrolladores" (CTRL+MAYUS+i):



Y ya se muestra la página web y el contenido de la consola (que usaremos para ver ciertos resultados de ejecución en JavaScript):



2.1.3. Ejecución del código en PHP y Python

En el caso de PHP y Python, el código no se ha de ejecutar en el navegador, si no en el servidor, por lo que es necesario hacer una configuración previa del servidor.

Aunque los detalles de instalación y configuración de servidores web se tratan en el módulo Despliegue de Aplicaciones Web, aquí vamos a hacer una pequeña guía para poder seguir los

ejercicios de este tema. En un despliegue profesional se usarían contenedores Dockers, pero por rapidez para iniciarnos en la programación, vamos a crear una máquina virtual en VirtualBox y sobre la misma vamos a instalar una versión de Ubuntu 22.04 LTS.

1. Creación de la máquina virtual para Ubuntu, asignándole RAM y disco suficientes.
2. Descarga e instalación de la imagen de Ubuntu 22.04 LTS.
3. Instalación de Ubuntu 22.04 LTS
4. Una vez instalado el sistema operativo y comprobado el funcionamiento del mismo, procedemos a la instalación del servidor web, para ello entramos en la línea de comando y ejecutamos el siguiente comando para actualice el sistema:

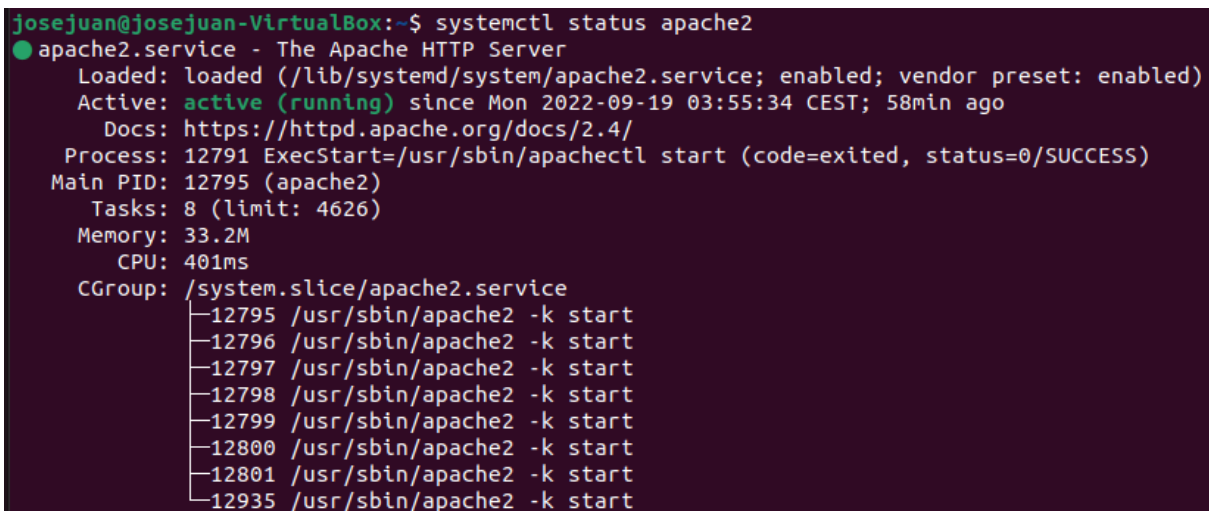
```
~$ sudo apt update
```

5. A continuación, instalaremos el servidor Apache y PHP:

```
~$ sudo apt install -y apache2 libapache2-mod-php
```

6. Podemos comprobar que el servidor está instalado y funcionando:

```
~$ systemctl status apache2
```



```
josejuan@josejuan-VirtualBox:~$ systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-09-19 03:55:34 CEST; 58min ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 12791 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
  Main PID: 12795 (apache2)
    Tasks: 8 (limit: 4626)
   Memory: 33.2M
      CPU: 401ms
   CGroup: /system.slice/apache2.service
           └─12795 /usr/sbin/apache2 -k start
             └─12796 /usr/sbin/apache2 -k start
               └─12797 /usr/sbin/apache2 -k start
                 └─12798 /usr/sbin/apache2 -k start
                   └─12799 /usr/sbin/apache2 -k start
                     └─12800 /usr/sbin/apache2 -k start
                       └─12801 /usr/sbin/apache2 -k start
                         └─12935 /usr/sbin/apache2 -k start
```

7. Editamos el fichero de configuración de PHP para cambiar dos aspectos que no son imprescindibles, pero sí recomendables. El fichero se encuentra en la siguiente ruta.

```
~$ sudo nano /etc/php/8.1/apache2/php.ini
```

8. En primer lugar vamos a cambiar la zona horaria del servidor, para lo que buscamos la directiva `date.timezone` (esta línea estará comentada con un `;` delante), la descomentamos y cambiamos el valor a `Europe/Madrid`

```
date.timezone = Europe/Madrid
```

9. A continuación vamos a cambiar la configuración sobre mostrar información de errores. Por defecto, PHP está configurado para un entorno de producción en el que, por motivos de seguridad, la información sobre los errores que se muestra es mínima (para evitar revelar información sensible y prevenir así posibles ataques). Pero para un entorno de desarrollo, es preferible que nos proporcione más información para facilitar la depuración de errores.

Localizamos las siguientes tres directivas:

```
error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT
```

```
display_errors = Off
```

```
display_startup_errors = Off
```

10. Y las cambiamos por los siguientes valores:

```
error_reporting = E_ALL
```

```
display_errors = On
```

```
display_startup_errors = On
```

11. Guardamos el archivo y reiniciamos el servicio de Apache

```
~$ sudo systemctl reload apache2
```

12. Sólo queda probar que todo funciona correctamente, en primer lugar abrimos un navegador y escribimos en la barra de direcciones `http://localhost` que es la ubicación donde se encuentra el servidor web:



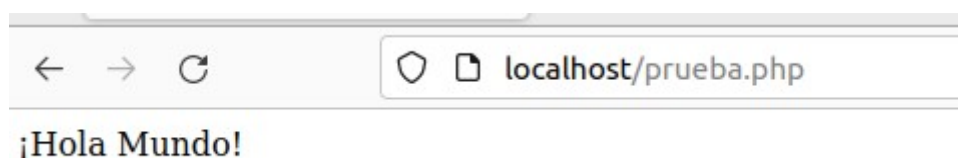
13. Ahora sólo queda comprobar que funciona PHP, para ello creamos un fichero en el directorio raíz de la página web:

```
~$ sudo nano /var/www/html/prueba.php
```

14. Y ponemos el siguiente código dentro del archivo

```
<?php echo "<h1>¡Hola Mundo!";">
```

A continuación, en el navegador, accediendo a la URL `localhost/prueba.php` nos debería mostrar:



En Python hay que hacer unos pasos adicionales. El motivo es que Python no es un lenguaje de script, no está integrado directamente dentro del HTML, así que debemos usar una de las pasarelas (middleware) de las que disponemos. Aunque no sea la más eficiente, vamos a usar CGI para conectar Apache a los programas que realicemos en Python. Ya se verá más adelante la instalación de middleware's más avanzados y eficientes cuando se vea un framework de Python.

1. En primer lugar abrimos un terminal de Ubuntu y comprobamos que tenemos instalado Python (debería venir por defecto en Ubuntu), una vez dentro de la interfaz de Python, simplemente salimos con `exit()`:

```
~$ python3

Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>exit()

~$ _
```

2. Si no estuviese instalado, lo instalamos y volveríamos a probar el paso 1:

```
~$ sudo apt -y install python3
```

3. A continuación activamos el módulo CGI de Apache:

```
~$ sudo a2enmod cgi
```

4. Y reiniciamos el servidor de Apache

```
~$ sudo systemctl reload apache2
```

5. A continuación vamos al directorio del raíz del sitio web, ubicado en `/var/www/html` y creamos una carpeta denominada `pyscripts` para contener los programas en Python:

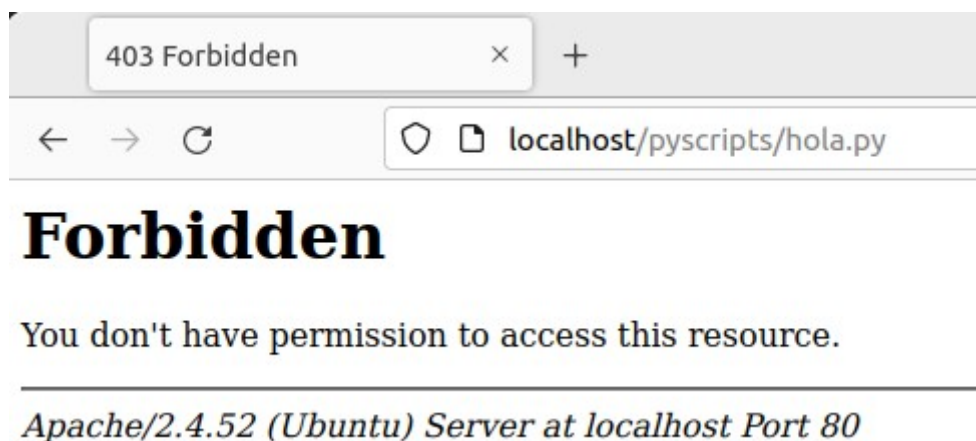
```
~$ cd /var/www/html
~$ sudo mkdir pyscripts
```

6. Dentro de esa carpeta, creamos un archivo denominado `hola.py` (puedes usar cualquier editor que consideres, incluso Visual Studio Code dentro de Ubuntu) y ponemos el siguiente contenido dentro (ya lo analizaremos más adelante):

```
~$ sudo nano hola.py
```

```
1    #!/usr/bin/python3
2    print("Content-type: text/html\n\n");
7. 3    print("Hola Mundo");
```

8. Si intentamos ejecutarlo, poniendo la URL, nos dará un error por falta de permisos:



9. En primer lugar, hay que darle permisos de ejecución al fichero (ya que es de hecho, un script de Python):

```
~$ sudo chmod 755 hola.py
```

10. Antes de poder ejecutar el script, también hay que decirle a Apache que cuando invoquemos un archivo con extensión .py debe ejecutarlo como un CGI, para eso hay que modificar dos ficheros. En primer lugar, el propio fichero de configuración de Apache:

```
~$ sudo nano /etc/apache2/apache2.conf
```

11. Con la directiva AddHandler le indicamos que los ficheros .py los trate como un CGI, al final del fichero de configuración añadimos la línea:

```
AddHandler cgi-script .py
```

12. Pero eso no es suficiente, ahora hemos de decirle en qué directorio vamos a permitir la ejecución de CGI's. Dentro de /etc/apache2 tenemos el directorio sites-enabled que indica los servidores virtuales activos. Dentro encontraremos un fichero con extensión .conf (si tenemos varios servidores, encontraremos varios archivos .conf). Editamos la configuración del sitio por defecto y añadimos las líneas correspondientes:

```
~$ cd /etc/apache2/sites-enabled
```

```
~$ sudo nano 000-default.conf
```

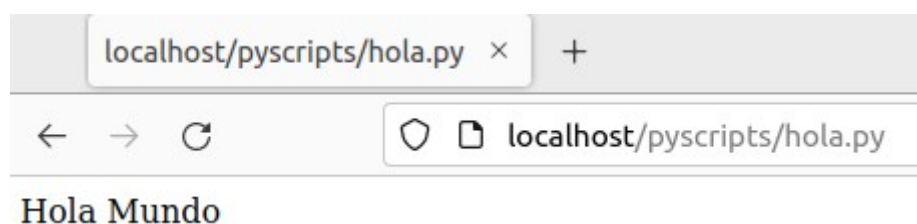
13. Encontramos una etiqueta `<VirtualHost *:80></VirtualHost>`, y dentro de la misma, la configuración del servidor. Añadimos al final (justo antes de `</VirtualHost>`) la activación de ejecución de scripts dentro de la carpeta que habíamos creado:

```
<Directory /var/www/html/pyscripts>  
    Options +ExecCGI  
</Directory>
```

14. De nuevo, reiniciamos el servidor de Apache

```
~$ sudo systemctl reload apache2
```

15. Y si ahora volvemos a intentar ejecutar el script, deberíamos ver lo siguiente:



2.2. Insertando código en JavaScript

Existen dos formas de integrar el código JavaScript dentro del HTML.

La forma más inmediata es mediante las etiquetas `<script>` `</script>` y empleando un atributo `type` indicaremos qué tipo de lenguaje de script estamos utilizando:

Por ejemplo:

```
<script type="text/javascript">  
//Aquí insertaríamos el código propiamente dicho  
</script>
```

La segunda forma de integrar el código de JavaScript es a través de un fichero externo que contenga el código de JavaScript. Usualmente esta segunda forma es preferible, ya que así se separa el código HTML del código JavaScript, de forma que una modificación en una de las partes

minimiza un error en la otra. Adicionalmente, es posible hacer reutilización de código. Si una función ha de usarse en varias páginas de nuestro sitio web, con incluir el archivo que la contiene, y tendríamos la función disponible. Esto proporciona modularidad al código y simplifica el mantenimiento (una modificación de la función no exige modificar todos los HTML que la contienen, sólo el fichero JavaScript). Para se ha de añadir a la etiqueta `script` el atributo `src`, con el nombre del fichero que contiene el código de JavaScript. Generalmente los ficheros que contienen texto de JavaScript tendrán la extensión `.js`.

Por ejemplo:

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

En caso de requerir incluir más de un fichero JavaScript, se podrán tantas etiquetas como sean necesarias.

Las etiquetas de `<script>` y `</script>` son obligatorias aunque se incluya un fichero externo. Si se escribe código entre ambas etiquetas y se usa a la vez el atributo `src`, dicho código no se ejecutará.

Los ficheros referenciados mediante `src` podrán encontrarse en la misma ubicación que el fichero HTML, en otros directorios (rutas relativa) e incluso en otros servidores (rutas absolutas), de forma análoga al atributo `src` de la etiqueta ``, siempre que se mantengan las convenciones de nombres de las URL's.

Ejemplos:

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

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

El código JavaScript se ha de descargar en el cliente, por lo que es un código que siempre será accesible por cualquier usuario que pueda acceder a la página HTML, cuestión que se ha de tener en cuenta desde el punto de vista de la seguridad (ej.: contraseñas de acceso, comentarios que puedan revelar aspectos sensibles del sistema, etc.). Esto también implica que no se puede

prevenir que terceros copien el código JavaScript

Puede ocurrir que el código se vaya a ejecutar en un navegador que no soporte JavaScript o lo tenga desactivado. Para se dispone de la etiqueta `<noscript>` contenido_alternativo `</noscript>` que permite indicar un contenido alternativo en caso de que el navegador no soporte JavaScript. Por ejemplo:

```
<script type="text/javascript">
    console.log("Texto generado desde JavaScript")
</script>
<noscript>
    <p> Su navegador no permite la ejecución de JavaScript</p>
</noscript>
```

2.3. Insertando código en PHP

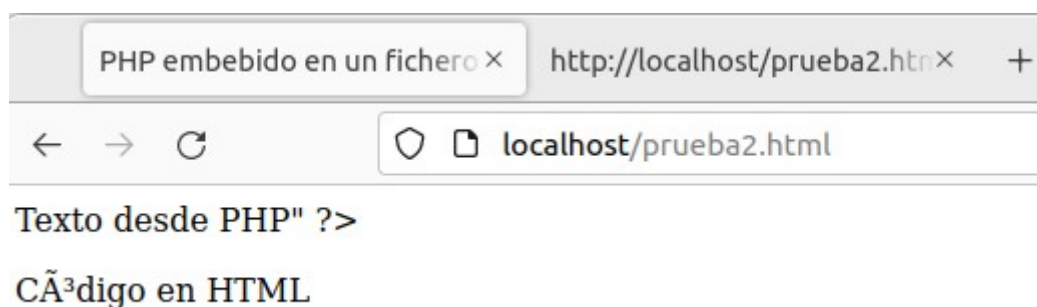
PHP se diseñó para combinar etiquetas HTML junto a código en JavaScript, por ese motivo, los ficheros con código PHP parecen una página HTML con código incrustado. Por ejemplo crearemos el siguiente archivo en el raíz de nuestro sitio web y le llamaremos `prueba2.html`:

```
1    <!DOCTYPE html>
2    <html>
3    <head>
4        <title>PHP embebido en un fichero .html</title>
5    </head>
6    <body>
7        <?php echo "<h1>Texto desde PHP</h1>" ?>
8        <p> Código en HTML</p>
9    </body>
10   </html>
```

Habrás podido comprobar como en la línea 7 aparece una etiqueta que no es HTML.

Con `<?php aquí_viene_el_código_en_php ?>` le indicamos al servidor que lo que está dentro de esa etiqueta no debe devolverlo al cliente, si no ejecutarlo e interpretarlo.

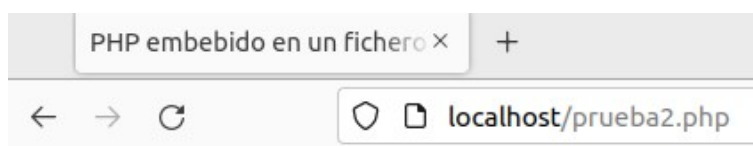
Si vamos al navegador y llamamos a dicha URL, obtendremos lo siguiente:



Y comprobamos que no ha funcionado como esperábamos, si revisamos el código fuente de la página (botón de la derecha, “Ver código fuente de la página”) nos muestra lo siguiente:



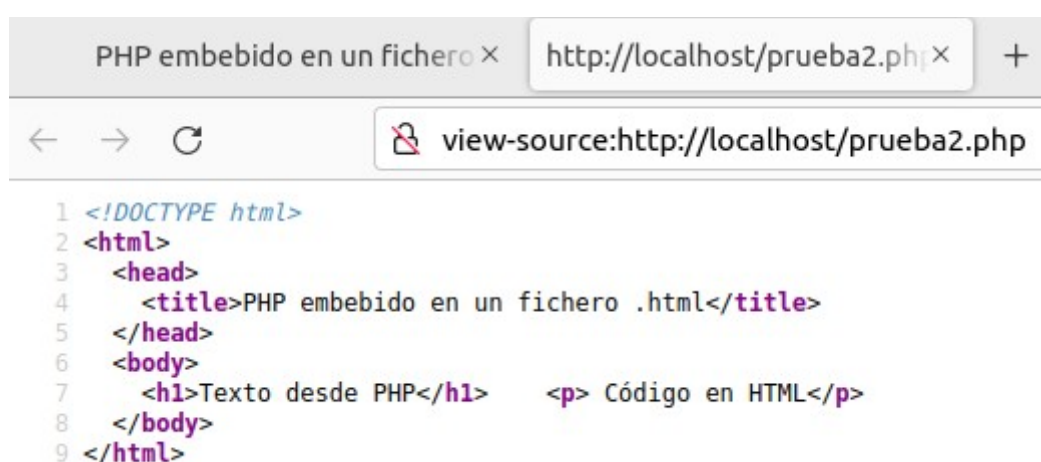
Y nos damos cuenta que el servidor no ha ejecutado el código, si no que lo ha devuelto directamente al cliente. El motivo es que es una página con extensión HTML, para que funcione correctamente, hemos de renombrar la página como prueba2.php, y ahora al volver a llamar a la nueva URL:



Texto desde PHP

Código en HTML

Y si analizamos el código fuente, ahora vemos lo siguiente:



El script ha generado el HTML correctamente.

2.4. Insertando código en Python

Python es un diferente a PHP, ya que Python es un lenguaje de programación genérico, aunque dada su efectividad y potencia es muy adecuado para su uso en la web. No obstante, si ejecutamos Python directamente, no podemos incrustarlo o embeberlo dentro de HTML. En un entorno real, en la práctica totalidad de las ocasiones se usará Python desde un framework como Django o Flask, pero a efectos de aprender el lenguaje y probar los ejemplos desde la web, vamos a ejecutar los scripts de Python directamente.

Para ello vamos a crear en el directorio pyscripts creado anteriormente un fichero denominado `ejemplo.py`. Ya hemos visto que la extensión `.py` es necesaria para que Apache sepa que es un

ejecutable (recuerda asignar permisos de ejecución con `chmod` al fichero para que se pueda ejecutar).

Dentro de ese archivo escribimos lo siguiente:

```
1  #!/usr/bin/python3
2
3  # Siempre hay que indicar el tipo de contenido en la cabecera http
4  print("Content-type: text/html\n\n");
5
6  # Se declara una variable y se construye la página HTML
7  salida = "<!DOCTYPE html>";
8  salida = salida+"<html lang='es'>";
9  salida = salida+"<head>";
10 salida = salida+"</head>";
11 salida = salida+"<body>";
12 salida = salida+"<h1>Texto desde Python</h1>";
13 salida = salida+"</body>";
14 salida = salida+"</html>";
15
16 # Se devuelve la página de salida al servidor web
17 print(salida);
```

En la primera línea encontramos lo siguiente:

```
#!/usr/bin/python3
```

Esta línea le dice a Apache que lo que viene a continuación ha de ejecutarlo con el programa que se indica, en este caso `python3`, que se encuentra en la ruta `/usr/bin`.

Ya veremos como más adelante esto no será necesario en el caso de los frameworks, pero por el momento deberemos incluirlo en todos los programas que hagamos.

En la siguiente línea se usa la función `print()` de Python de impresión por la salida estándar para devolver la cadena `"Content-type: text/html\n\n"`. Como estamos usando el estándar CGI, en realidad el programa debe encargarse de devolver toda la respuesta HTTP, ya que Apache no lo hará por nosotros, así que hemos de incluir como mínimo la cabecera `Content-type`, si no fallará la devolución de datos. Es importante incluir los dos saltos de línea `"\n\n"` ya

que así está definido en el estándar HTTP.

A continuación se usa una variable denominada “`salida`” para ir construyendo la respuesta que vamos a enviar al cliente. Podrás darte cuenta que todo el código HTML que se ha de devolver hemos de incluirlo como cadenas dentro del lenguaje. No podemos usar código HTML directamente dentro del código porque nos daría un error.

Dos cosas que es importante reseñar:

1. No hemos declarado explícitamente la variable, simplemente la hemos empezado a usar.
2. Tampoco hemos indicado de qué tipo es la variable, directamente le hemos asignado una cadena de texto. Más adelante se tratarán en profundidad los tipos de datos de Python.

Por último, una vez concatenada toda la respuesta que le vamos a enviar al cliente, la devolvemos de nuevo con la función `print()`.

3. COMENTARIOS EN EL CÓDIGO

Cada uno de los lenguajes usa un sistema para incluir comentarios en el código. Recuerda la importancia de documentar el código, ya que suele ser una tarea que los programadores siempre dejan para el final pensando “Ya documentaré el código cuando acabe”, aunque la realidad es que cuando se finaliza un programa, en seguida suele ser necesario comenzar con otro y se acaba abandonando la tarea de documentar. Sin embargo, documentar el código, aunque no lo parezca al principio, puede agilizar el desarrollo del software. Algunos motivos para documentar son los siguientes:

1. Nos permite comprender nuestro propio código

Un código, aunque lo hayamos programado nosotros mismos, pasados unos días o semanas nos llega a parecer extraño y más de una vez nos hemos preguntado “¿qué es lo que yo pretendía hacer aquí?” Si el código lleva escrito mucho más tiempo (meses o años) es un auténtico desafío averiguar qué queríamos hacer en aquel momento y porqué se programó como se programó. Una buena documentación evitado pasar mucho tiempo descifrando nuestro propio código.

2. Ayuda a otros

Si entender el código propio hecho hace un tiempo, esto se agudiza si hemos de compartir el código con otros, que es lo que usualmente se hace en las empresas cuando se trabaja en equipo. No son necesarias interrupciones para preguntarte cómo o porqué se hizo el programa. Simplemente leyendo la documentación se ahorra tiempo para todo el equipo.

3. Ayuda a corregir errores fácilmente

En el caso de necesitar corregir o modificar una parte del programa, no es necesario buscar por todo el código dónde se realiza tal o cual cosa. Es fácil localizar exactamente el

trozo de código buscado.

4. Mantiene claro el objetivo

En ocasiones, se encuentran trozos de código que no tienen, aparentemente, ninguna función y no se recuerda por qué está eso ahí ¿quizás era una prueba? ¿Algo que se usa en un caso concreto o en una excepción? ¿se puede borrar o se ha de mantener? Si es un código importante o solo una prueba, la documentación lo habría indicado.

5. Es posible reutilizar el código

Es bastante frecuente que necesitemos hacer algo que anteriormente ya habíamos hecho. En lugar de recorrer todo nuestro código buscando aquella función que creemos recordar hacia lo que necesito, es más fácil y rápido consultar la documentación, ver si existe y localizar el código buscado.

Una vez repasada la importancia de documentar, vamos a ver cómo se hace en cada uno de los lenguajes que vamos a ver en el curso.

3.1. Comentarios en JavaScript

JavaScript tiene dos opciones para poner comentarios, comentarios de una línea, que comienzan con doble barra `//` y los comentarios multi línea, que al estilo C empiezan con `/*` y deben finalizar con `*/`. Ejemplos:

```
// Este es un comentario de una línea

/*
    Este es un comentario que usa múltiples líneas.
    Incluso es posible comentar
    bloques enteros de código
*/
```


3.2. Comentarios en PHP

Los comentarios en PHP son similares a los de JavaScript, incluyendo doble barra `//` para comentarios de una línea y `/* aquí viene el comentario */` para los comentarios multilínea.

Adicionalmente, añade también otra posibilidad de comentarios para una línea, con el símbolo `#`, que funciona exactamente igual que la doble barra:

```
// Este es un comentario de una línea  
  
# Este también es un comentario de una línea
```

Una cuestión importante es que los comentarios no se envían al cliente, ya que podría suponer un riesgo de seguridad.

3.3. Comentarios en Python

En python los comentarios se pueden poner de dos formas:

1. Con el símbolo almohadilla `#` para los comentarios de una única línea.
2. Con triple comilla doble `"""` al principio y al final de los comentario multilínea.

Por ejemplo:

```
# Comentario de una línea en Python  
  
"""  
    Comentario multilínea,  
    que al igual que en otros lenguajes  
    también podría usarse para comentar código  
    """
```

4. VARIABLES

Una variable no es más que un espacio de almacenamiento en la memoria principal, espacio al que se le ha asignado un nombre simbólico, o identificador, asociado a dicho espacio. Ese espacio alberga una cantidad de información (de tamaño variable según la naturaleza de la información a tratar) a la que denominamos valor. El valor será conocido en tiempo de programación, por ejemplo si asignamos directamente un valor, o desconocido, si es un valor que se pasará en tiempo de ejecución. El nombre de la variable es la forma usual de referirse al valor almacenado. La separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa.

4.1. Variables en JavaScript

En JavaScript, una variable es cualquier cadena alfanumérica que cumpla las siguientes reglas:

- No podremos usar palabras reservadas del propio lenguaje (for, if, etc.)
- No se pueden usar símbolos de puntuación en el medio de la variable
- Una variable no podrá contener espacios en blanco.
- Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_).
- No podremos utilizar caracteres como el signo +, %, etc. en los nombres de variables
- No pueden comenzar con un número.
- Los nombres de variable son *case-sensitive*.

Se puede usar una variable sin haberla declarado anteriormente, como por ejemplo:

```
a=5;
```

Pero esto no se recomienda, y por ello, disponemos de las palabras reservadas: var, let y const.

Var y let

Se puede declarar una variable usando la palabra reservada let (también se puede usar var, pero no se recomienda desde la versión ES2015). En JavaScript las variables no tienen un tipo de dato

explícito. El tipo puede cambiar internamente dependiendo de cual sea el valor que se le asigne.

Esto significa que puedes asignar un string, y posteriormente un número.

```
let v1 = "Hola Mundo!";  
  
console.log(typeof v1); // Imprime -> string  
  
v1 = 123;  
  
console.log(typeof v1); // Imprime -> number
```

Cuando se declara una variable, pero no se le asigna un valor, tendrá un tipo especial conocido como `undefined`. Nota: Este valor es diferente de `null` (al cual, en JavaScript, se considera un valor).

```
let v1;  
  
console.log(typeof v1); // Imprime -> undefined  
  
if (v1 === undefined) {  
    console.log("Has olvidado darle valor a v1");  
}
```

Si una variable no la declaramos, o la hacemos con `var`, JavaScript declarará esa variable como global. Esto no es lo recomendado porque las variables globales son peligrosas. Por tanto, es recomendable que usemos siempre `let`, ya que la variable se circunscribe al ámbito (scope) en el que se declara. El scope puede definirse como el alcance que una variable tendrá en el código. A efectos prácticos el scope será el código que se encuentre entre dos llaves `{ }`. En otras palabras, el scope decide a qué variables tienes acceso en cada parte del código. Existen dos tipos de scope, el scope global y el scope local.

Para evitar que se nos olvide declarar con `let`, podemos usar una declaración especial: `"use strict"` al comienzo de nuestro archivo JavaScript, y de hecho es una buena práctica de programación hacerlo siempre. De este modo, no es posible declarar variables globales omitiendo la palabra reservada `let`.

```
'use strict';  
  
v1 = "Hola Mundo"; //Este código da un error
```

Const

Cuando a lo largo de una función o bloque, una variable no va a cambiar de valor, o cuando queremos definir un valor global inmutable (por ejemplo el número PI), se recomienda declararla como constante con la palabra reservada `const` en lugar de usar `let`. En el caso de las constantes globales se recomienda usar mayúsculas.

```
'use strict'  
  
const MY_CONST=10; //Correcto, se le da valor cuando se declara  
MY_CONST=200; //Error, ya no se le puede asignar otro valor
```

4.2. Variables en PHP

En PHP las variables tienen que seguir las siguientes reglas:

- Se representan con un signo de dólar \$ seguido por el nombre de la variable.
- El nombre de la variable es sensible a minúsculas y mayúsculas.
- Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado (underscore), seguido de cualquier número de letras, números y caracteres de subrayado.
- Se podría expresar como: `'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'` en una expresión regular. una letra es a-z, A-Z, y los bytes del 127 al 255 (0x7f-0xff).
- `$this` es una variable especial que no puede ser asignada

```
<?php  
  
$var = 'Roberto';  
  
$Var = 'Juan';  
  
echo "$var, $Var";          // imprime "Roberto, Juan"
```

```
$4site = 'aun no';          // inválido; comienza con un número  
$_4site = 'aun no';        // válido; comienza con un carácter de subrayado  
$täyte = 'mansikka';       // válido; 'ä' es ASCII (Extendido) 228  
?>
```

De forma predeterminada, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor completo de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después de asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra.

PHP también ofrece otra forma de asignar valores a las variables: asignar por referencia. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" ó "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Para asignar por referencia, simplemente se antepone un signo ampersand (&) al comienzo de la variable cuyo valor se está asignando:

```
<?php  
$varA = 1; //A la variable varA se le asigna un valor  
$varB = &$a; //La variable varB apunta a varA  
$varA = 2;  
echo $varB //Muestra 2, aunque no se le haya asignado expresamente  
?>
```

No es necesario inicializar variables en PHP, sin embargo, es una muy buena práctica. Las variables no inicializadas tienen un valor predeterminado de acuerdo a su tipo dependiendo del contexto en el que son usadas - las booleanas se asumen como false, los enteros y flotantes como cero, las cadenas (p.ej. usadas en echo) se establecen como una cadena vacía y los arrays se convierten en un array vacío.

Las variables en PHP también tienen sus ámbitos de alcance, pero a diferencia de otros

lenguajes, las variables inicializadas en ámbitos superiores no se encuentran disponibles en ámbitos inferiores. Si necesitamos hacer esto, tendremos que usar la directiva global cuando se asigne valor a la variable la primera vez.

Por ejemplo:

```
<?php
$varA = 1; // ámbito externo

function test() {
    //No muestra nada, PHP interpreta que es otra variable
    echo $varA;
}

test();
?>
```

Para solucionar eso, usamos global:

```
<?php
$varA = 1; // ámbito externo

function test() {
    //Ahora identifica la variable interna con la externa
    global $varA;
    echo $varA;
}

test();
?>
```

4.3. Variables en Python

Python no tiene un comando para declarar variables. Las variables se crean en el momento en que se les asigna un valor la primera vez.

```
x = 5
```

```
y = "Juan"

print(x) # muestra 5 por pantalla

print(y) # muestra Juan por pantalla
```

Las reglas para declarar variables son:

- Cada variable tiene que tener un nombre con el que referirnos a ella.
- Python también es *case sensitive*, es decir, tiene en cuenta si escribimos en mayúsculas o minúsculas la variable. Por ejemplo, las variables dato, Dato, DaTo y DATO son distintas entre ellas.
- El nombre de la variable no puede coincidir con las palabras reservadas del lenguaje (if, for, etc.). Tampoco podremos usar nombres de variables con tildes o con ñ.
- Pueden empezar por letras y guiones bajos '_' pero nunca pueden empezar por números.

Es decir `_variable_1` es una variable correcta, pero `1_variable` no lo es.

Si se usa una variable a la que no se le ha asignado un error, Python da un error.

En Python las variables de ámbitos superiores sí están disponibles en ámbitos inferiores, a menos que se asigne valor a una variable, que la definirá interna. Por ejemplo, en el siguiente código:

```
6   a=1;
7   def f():
8       print(a);
9
10  f();
```

Dentro de la función `f()` la variable `a` está perfectamente definida y se muestra por pantalla su valor. Sin embargo, en el siguiente código se produce un error, e indica que en la línea 8 se ha accedido a una variable no inicializada:

```
6   a=1;
7   def f():
8       print(a);
9       a = a+1;
10
11  f();
```

```
File "/var/www/html/pyscripts/./hola.py", line 8, in f
    print(a);
UnboundLocalError: local variable 'a' referenced before assignment
```

¿Por qué se ha producido ese error? Si Python crea una variable nueva dentro de la función al asignarle valor, parece que eso debería ocurrir en la línea 9, por lo tanto, en la línea 8 aún debería tener el valor del ámbito externo. Sin embargo Python, cuando entra en un ámbito, hace un análisis previo y encuentra que se hará una declaración a la variable `a`, y hace (por así decirlo) una reserva de ese nombre en ese ámbito. Cuando ejecuta posteriormente el `print` de la línea 8, busca la referencia de la variable `'a'` y la encuentra en el ámbito interno, pero sin asignar, y por eso da un error.

Hay que indicar que JavaScript también hace una extensión de ámbito de las variables de forma parecida a como hace Python.

La forma de solucionar el problema anterior, es indicar con `global` que nos referimos a la variable externa.

```
6   a=1;
7   def f():
8       global a;
9       print(a);
10      a=a+1;
11
12  f();
```