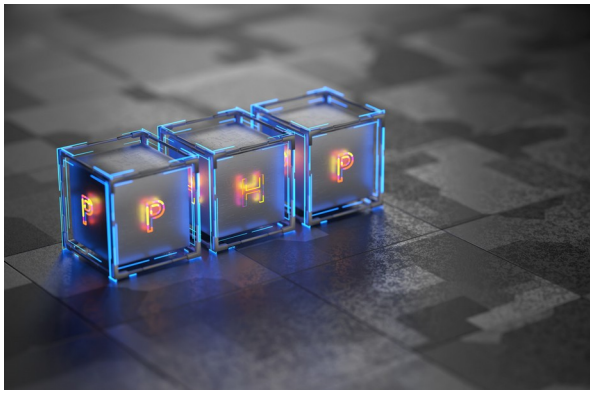
Desarrollo web en entorno servidor

Unidad 3: Programación basada en lenguajes de marcas con código embebido

ÍNDICE

INTRODUCCIÓN



En la unidad 1 hemos visto distintos **entornos de desarrollo** y hemos seleccionado el que mejor se nos adapta. Después, en la unidad 2, hemos analizado los distintos **lenguajes de programación** en el entorno de servidor y hemos escogido PHP para que nos acompañe a lo largo de todo el módulo.

Ahora ya podemos empezar a **aprender la sintaxis de PHP** para que puedas empezar a crear aplicaciones con una lógica potente, más allá de mostrar unos pocos textos por pantalla.

Por ello implicaremos el uso de tipos de datos complejos, como son los **arrays**, y bloques de código, ya no sólo estético sino funcionales mediante funciones y métodos.

También aprenderemos a recoger y mandar datos de y hacía el cliente con las **variables** y **métodos GET** y **POST**.

Aunque utilicemos la **sintaxis de PHP**, los conceptos que aquí aprenderás te servirán también si en un futuro utilizas otros lenguajes de programación como JSP o ASP.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Utilizar lenguajes, objetos y herramientas, interpretando las especificaciones para desarrollar aplicaciones web con acceso a bases de datos.**
- ✔ **Utilizar herramientas y lenguajes específicos, cumpliendo las especificaciones, para desarrollar e integrar componentes software en el entorno del servidor web.**
- ✔ **Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Rendimiento de un empleado

En la actividad de evaluación de la unidad 2 hicimos un *script* para saber si un empleado trabajaba más o menos de sus 38 horas semanales. Ahora mejoraremos ese *script* ya que haremos un cálculo del salario semanal en función de su rendimiento.

- Su salario/hora es de 17€ brutos, con lo cual, si hiciera las 38 horas como estipula el contrato el empleado cobraría 646€ a la semana.
- Si el empleado hace menos horas, cobrará proporcionalmente su salario, es decir, que, si hace sólo 20 horas, cobrará 340€.
- En cambio, si hace horas extra, cada hora extra la cobrará al doble de su precio. Por ejemplo, si trabaja 40 horas cobrará (38 x 17) + (2 x 34), es decir, 714€. El tope máximo de horas extra pagadas será de 10 horas. Si el trabajador hace más de 10 horas extra no deberán tener en cuenta.

Guarda los siguientes datos, con un salto de línea entre ellos en el fichero `juan.txt`:

8 9 7 10 8

Estos datos representan las horas que el trabajador Juan ha realizado a lo largo de una semana.

Crea un formulario que recoja el nombre del fichero. Posteriormente recupera los datos de este archivo y calcula el sueldo que la empresa debería pagarle al trabajador.

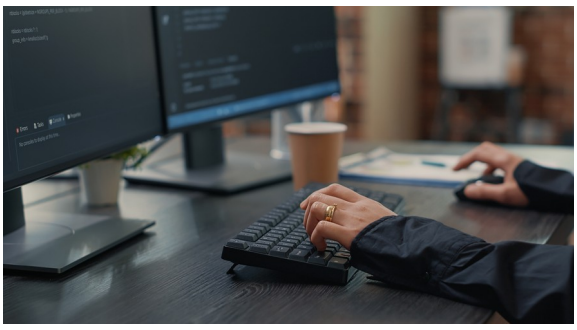
1. TOMAS DE DECISIÓN

Hasta ahora hemos visto programas con una estructura **secuencial**, es decir el programa se ejecuta de principio a fin, ejecutando todas las instrucciones.

*Muy a menudo, cuando escribe código, se desean realizar diferentes acciones para diferentes condiciones. Estas tomas de decisión son las **declaraciones condicionales**.*

En PHP tenemos las siguientes **declaraciones condicionales**:

- `if` declaración: ejecuta algún código si una condición es verdadera.
- `if...else` declaración: ejecuta algún código si una condición es verdadera y otro código si esa condición es falsa.
- `if...elseif...else` declaración: ejecuta diferentes códigos para más de dos condiciones.
- `switch` instrucción: selecciona uno de los muchos bloques de código que se ejecutarán.



➔ **Sintaxis if**.

```
if (condición) {
  bloque de código a ejecutar si se cumple la condición;
}
```

✔ **Ejemplo if**:

```
$edad = 20;
if ($edad >= 18) {
  echo "Ya eres mayor de edad";
}
```

- ✔ **Resultado if**: ya eres mayor de edad.

➔ **Sintaxis if...else**.

```
if (condición) {
  bloque de código a ejecutar si se cumple la condición;
} else {
  bloque de código a ejecutar si no se cumple la condición;
}
```

- ✔ **Ejemplo if...else**:

```
$edad = 20;
if ($edad >= 18) {
  echo "Ya eres mayor de edad";
} else {
  echo "Aún eres menor de edad";
}
```

- ✔ **Resultado if...else**: ya eres mayor de edad.

➔ **Sintaxis if...elseif...else**.

```
if (condición) {
  bloque de código a ejecutar si se cumple la condición;
} elseif (condición) {
  bloque de código a ejecutar si no se cumple la primera condición, pero si se cumple esta;
} else {
  bloque de código a ejecutar si no se cumplen las demás condiciones;
}
```

- ✔ **Ejemplo if...elseif...else**:

```
$edad = 20;
if ($edad < 18) {
  echo "Aún eres menor de edad";
} elseif ($edad < 65) {
  echo "Ya eres mayor de edad";
} else {
  echo "Ya tienes la edad de jubilación";
}
```

- ✔ **Resultado if...elseif...else**: ya eres mayor de edad.

➔ **Sintaxis switch**.

```
switch (n) {
  case valor1:
    bloque a ejecutar si n=valor1;
    break;
  case valor2:
    bloque a ejecutar si n=valor2;
    break;
  ...
  default:
    bloque a ejecutar si n no coincide con ningún valor;
}
```

- **Ejemplo switch**:

```
$dia = "martes";
switch ($dia) {
  case "sábado":
    echo "¡Te espera un largo fin de semana!";
    break;
  case "domingo":
    echo "¡Aprovecha, que mañana es lunes";
    break;
  default:
    echo "¡Toca ir a trabajar!";
}
```

- **Resultado switch**: ¡toca ir a trabajar!

2. BUCLES

A veces, cuando se programa, se desea que el mismo bloque de código se ejecute una y otra vez una determinada cantidad de veces. Entonces, en lugar de agregar varias líneas de código casi iguales en un *script*, podemos usar bucles.

> *Los **bucles** se utilizan para ejecutar el mismo bloque de código una y otra vez, siempre que se cumpla una determinada condición.*

En PHP, tenemos los siguientes tipos de bucle:

- `while`: recorre un bloque de código siempre que la condición especificada sea verdadera.
- `do...while`: recorre un bloque de código una vez y luego repite el ciclo siempre que la condición especificada sea verdadera.
- `for`: recorre un bloque de código un número específico de veces.
- `foreach`: recorre un bloque de código para cada elemento en una matriz.

➔ **Sintaxis while**.

```
while (si se cumple la condición) {
  bloque de código a ejecutar;
}
```

✔ **Ejemplo while**:

```
$x = 1;
while($x < 3) {
  echo "El número es: $x <br>";
  $x++;
}
```

✔ **Resultado while**:

- El número es: 1.
- El número es: 2.

➔ **Sintaxis do...while**.

```
do {
  bloque de código a ejecutar;
} while (si se cumple la condición);
```

✔ **Ejemplo do...while**:

```
$x = 5;
do {
  echo "El número es: $x <br>";
  $x++;
} while($x < 3);
```

✔ **Resultado do...while**:

- El número es: 5.

➔ **Sintaxis for**.

```
for (valor inicial; comprobación del valor; incremento del valor) {
  bloque de código a ejecutar en cada iteración;
}
```

✔ **Ejemplo for**:

```
for ($x = 0; $x < 100; $x++) {
  echo "$x <br>";
}
```

✔ **Resultado for**:

- Muestra los números de 0 a 99.

➔ **Sintaxis foreach**.

```
foreach ($array as $valor) {
  bloque de código a ejecutar por cada valor del array;
}
```

✔ **Ejemplo foreach**:

```
$semana = array("lunes", "martes",
"miércoles", "jueves", "viernes",
"sábado", "domingo");

foreach ($semana as $dia) {
  echo "$dia <br>";
}
```

✔ **Resultado foreach**:

- Muestra los días de la semana.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cuál de las siguientes estructuras condicionales es correcta?

a. `if … else … elseif`.

b. `if … elseif … else`.

c. `else … if … elseif`.

Lee el enunciado e indica la opción correcta:

¿Cuál de estas condiciones da como resultado true?

a. `3 / 2 >= 1.5`.

b. `3 / 2 = 1`.

c. `3 / 2 === "1.5"`.

Lee el enunciado e indica la opción correcta:

¿Cuántas veces como mínimo se ejecutará un bucle?

a. 1 o más.

b. 2 o más.

c. 0 o más.

3. COMENTARIOS DE CLIENTES Y DE SERVIDOR

*Un **comentario en el código de PHP** es una línea que no se ejecuta como parte del programa. Su único propósito es que alguien que esté mirando el código lo lea.*

Los **comentarios** se pueden utilizar para:

- Dejar que otros programadores **entiendan el código**.
- **Recordar lo que se hizo**: a veces la propia sintaxis del código es compleja y un comentario puede ayudar a entender lo que se hizo.

PHP admite varias **formas de comentar**:

➔ Comentarios **de una sola línea**.

Ejemplo:

```
<?php
// Esto es un comentario de una sola línea

# Esto también es un comentario de una sola línea
?>
```

➔ Comentarios **de varias líneas**.

Ejemplo:

```php
<?php
/*
Este es un comentario
que ocupa un bloque
de varias líneas
*/
?>
```

➔ Comentarios **para omitir parte del código**.

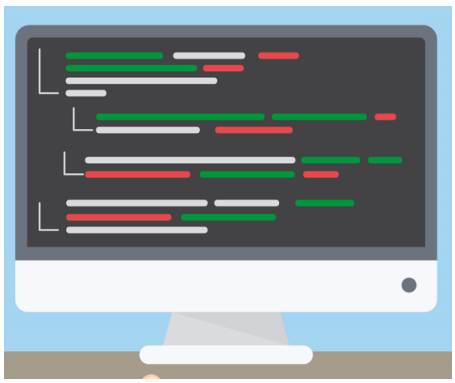
Ejemplo:

```php
<?php
// Si es preciso se puede comentar un fragmento de código
$x = 5 /* + 15 */ + 5;
echo $x;
?>
```

4. TIPOS DE DATOS COMPUESTOS

En la UD2 ya vimos que en PHP existen los siguientes **tipos de datos compuestos**:

- Array.
- Object.
- Callable.
- Iterable.



Vamos a definir cada uno de ellos y ver un **ejemplo**.

➔ **Definición de array**.

Un array, o matriz, es una **estructura de datos** para almacenar muchos valores dentro de una sola variable.

✔ **Ejemplo de array**:

```
<?php
$comida = array("Macarrones", "Tortilla", "Albóndigas");
echo "Mis platos favoritos son " . $comida[0] . ", " . $comida[1] . " y "
. $comida[2] . ".";
?>
```

✔ **Resultado de array**: mis platos favoritos son macarrones, tortilla y albóndigas.

➔ **Definición de object**.

Las clases y los objetos son los dos aspectos principales de la programación orientada a objetos.

Una **clase** es una plantilla para objetos y un **objeto** es una instancia de una clase.

Este es un tipo de datos que se estudiará en profundidad más adelante.

- ✔ **Ejemplo de object**:

```
<?php
class Animal {
    public $tipo;
    public $nombre;

    public function __construct ($tipo, $nombre){
        $this->tipo = $tipo;
        $this->nombre = $nombre;
    }

    public function hablar() {
      switch ($this->tipo) {
            case ("perro"):
                 echo $this->nombre . " hace: 'Guau'";
                 break;
             case ("gato"):
                 echo $this->nombre . " hace: 'Miau'";
                 break;
             default:
                 echo "Este animal no habla";
        }
    }
}

$animal1 = new Animal("perro", "Rufus");
$animal1->hablar();
?>
```

- ✔ **Resultado de object**: Rufus hace: 'guau'.

➔ **Definición de callable**.

El tipo de dato **callable** sirve para evidenciar que una variable realmente es una función. Esta característica da sentido a los **callbacks**.

✔ **Ejemplo de callable**:

```
<?php
function mostrar(callable $formato, $str) {
  echo $formato($str);
}

function exclamar($str) {
  return "¡" . $str . "!";
}

mostrar("exclamar", "Hola Mundo");
?>
```

✔ **Resultado de callable**: ¡Hola Mundo!

➔ **Definición de iterable**.

Iterable es un pseudotipo introducido en PHP 7.1. Acepta cualquier array u objeto que implemente la **interfaz traversable**. Estos dos tipos se recorren con foreach.

✔ **Ejemplo de iterable**:

```php
<?php
function show(iterable $iterable) {
    foreach ($iterable as $valor) {
        echo "$valor <br/>";
    }
}
$semana = ["lunes", "martes", "miércoles", "jueves", "viernes", "sábado",
"domingo"];
show($semanaArray);
?>
```

- ✔ **Resultado de iterable**: los días de la semana, una bajo el otro.

5. ARRAYS

*Un **array** es un tipo de variable que puede contener más de un valor en su interior Es muy útil para agrupar conjuntos de elementos usuarios, vehículos, clientes, etc.*

En PHP encontramos **3 tipos de arrays**:

- Arrays **indexados** [accedemos a sus valores a través de un índice].
- Arrays **asociativos** [accedemos a sus valores con una clave (clave-valor)].
- Arrays **multidimensionales** [*arrays* que contienen uno o más *arrays*].

El array es una de las **estructuras de datos** más utilizadas. Es muy conveniente estar familiarizado en su uso y sobre todo cómo recorrer y acceder a sus elementos.

➔ Ejemplo de un **array indexado** en el que sólo encontramos valores. Estos valores pueden ser de cualquier tipo *strings, integers, boolean*, etc., o incluso mezclados:

```
$arr = array("Peter",true,666);
var_dump($arr[0]);   // Ejemplo de acceso a una posición concreta
foreach ($arr as $value) {
     var_dump($value);
}
```

Fíjate que para construir el array hemos usado `array()` y se lo hemos asignado a una variable.

➔ En un **array asociativo** usualmente utilizamos un *string* como clave, pero no tiene por qué ser así:

```
$arr = array(
  1 => "Juan",
  "defensa" => "Ana",
  20 => "Carmen",
  true => "Pedro"
);
foreach ($arr as $value) {
  var_dump($value);
}
```

En el ejemplo anterior el nombre de Juan no aparece cuando ejecutamos el código, ya que el índice de Juan y Pedro es el mismo, aunque este último sea true (que simboliza 1), y se sobrescriben.

➔ Lo último que nos falta por ver son los **arrays multidimensionales**. Este tipo de *arrays* son muy comunes a la hora de intercambiar datos a nivel informático, sobre todo encapsulados en **formato JSON**, por lo tanto, a lo largo de tu carrera tendrás que enfrentarte a menudo. Un pequeño ejemplo para ver cómo funcionan con PHP:

```
$arr = array(
     "platos" => array(
       "primeros" => array("Zumo de tomate", "Ensalada"),
       "segundos" => array("Rabo de toro", "Merluza"),
        "postres" => array("Fruta", "Helado de chocolate")
     )
);
echo $arr["platos"]["segundos"][1]; // Merluza
```

Si quisiéramos recorrer este *array* podríamos ir uniendo `foreach()`. Pero si no conocemos la estructura ni el tamaño del array deberíamos utilizar **funciones recursivas** basadas en la teoría de grafos, temática que se escapa de este curso.

➔ **Tres tipos de arrays**

Vídeo: estructuras vistas: variables, operadores, sentencias de control, bucles y arrays

Visualiza el siguiente vídeo en el que verás las distintas estructuras que se han visto a lo largo de la unidad, como variables, sentencias de control, bucles, etc.

Actividad de aprendizaje 1: estructuras de control y bucles

Dado el siguiente array, debes recorrerlo y mostrar por pantalla aquellos días en los que la temperatura ha sido superior a 20 grados:

```
$semana = array(
  "lunes" => 18,
  "martes" => 19,
  "miércoles" => 22,
  "jueves" => 20,
  "viernes" => 17,
  "sábado" => 23,
  "domingo" => 24,
);
```

Crea una visualización atractiva de los datos y comparte en el foro el resultado, pero no el código fuente.

6. FUNCIONES. PARÁMETROS

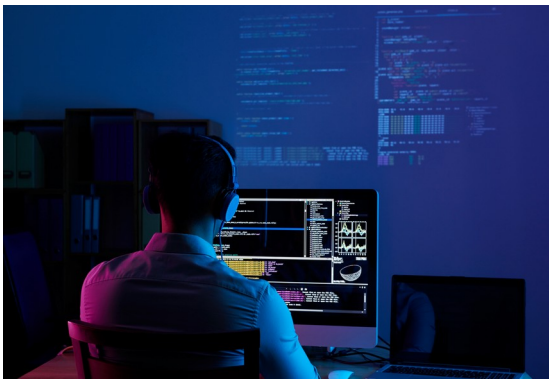
DESTACADO

*Una **función** es un blog de código que puede ser usado una o más veces en cualquier punto de nuestro código. Es una buena forma de tener código **limpio**, **estructurado** y **escalable**. PHP tiene más de 1000 funciones implícitas, algunas ya las hemos visto, pero también podemos crear otras nuevas.*

A continuación, veremos distintas **características** de una función:

- Una función en PHP tiene este **aspecto**:

```
function writeMsg() {
    echo "Hello world!";
}
writeMsg();  // llama a la función
```



- Además, una función puede tener **parámetros o argumentos**:

```
function writeMsg($msg) {
    echo $msg;
}
writeMsg("Hello world");
writeMsg("Today is a beautiful day");
```

- Las funciones pueden tener **tantos parámetros como sea necesario**. Los **parámetros son variables** (del tipo que queramos). Si una función tiene, por ejemplo 3 parámetros, debemos llamarla pasándole estos 3 parámetros, a menos que le asignemos unos valores por defecto:

```
function writeMsg($name="John", $surname="Doe") {
    echo "Hello $name $surname";
}
writeMsg();
writeMsg("James");
writeMsg("James", "Bond");
```

➔ Las funciones también pueden **devolver valores**:

```
function suma($a, $b) {
    return ($a + $b);
}
echo suma(5,6);
```

➔ Las funciones que has visto hasta ahora tenían nombre (writeMsg, suma, etc.) por tanto tienen una forma regular normal, o digamos, nominales. Pero también **existen funciones anónimas** que son aquellas que carecen de nombre:

```
function () {
    return "Hola";
}
```

➔ Las **funciones anónimas** se llaman de dos formas:

- ✔ Llamarla desde otra función en forma de **argumento**: visto en el apartado 4 al tratar los tipos de datos compuestos (*callable*).
- ✔ Con la sintaxis de **funciones variables**:

```
$saludo = function() {
    return "Hola!";
};
echo $saludo(); // Hola!
```

Cuestionario

Lee el enunciado e indica la opción correcta:

La estructura de datos que nos permite guardar varios valores en una sola variable se llama…

a. Array.

b. Booleano.

c. String.

Lee el enunciado e indica la opción correcta:

En un array asociativo accedemos a sus valores con:

a. El índice.

b. La clave.

c. Una combinación de clave e índice.

Lee el enunciado e indica la opción correcta:

¿Es correcta esta expresión? `$x = 5 /* + 15 */ + 5;`

a. No, porque para los comentarios en una sola línea deben usarse `(//)` o `(#)`.

b. Sí, es correcta.

c. No, no se puede trocear el código.

7. PASO DE PARÁMETROS. DEVOLUCIÓN DE VALORES

En el apartado anterior ya hemos visto que las funciones **reciben y devuelven valores**, pero nos ha faltado explicar que en las funciones **anónimas** pasa exactamente lo mismo.

➔ **Funciones anónimas (I)**:

Las **funciones anónimas**, que también se llaman cierres o *closures*, pueden recibir parámetros. Siguiendo el ejemplo `$saludo` del apartado anterior, tendríamos:

```
$saludo = function($nombre){
    echo "Hola " . $nombre;
};
$saludo("Juan");
$saludo("Ana");
```

➔ **Funciones anónimas (II)**:

Ahora probamos los siguientes dos códigos que parecen **iguales** (se ha marcado la diferencia en mayúscula) y nos fijamos qué pasa cuando los **ejecutamos**.

✔ Código **uno**:

```
$nombre = "Pepe";
$saludo = function() {
    echo "Hola " . $nombre;
};
$saludo();
```

✔ Código **dos**:

```
$nombre = "Pepe";
$saludo = function() USE ($NOMBRE) {
    echo "Hola " . $nombre;
};
$saludo();
```

*En el primer ejemplo, desde dentro de la función, no tenemos acceso a la variable `$nombre`. En el segundo ejemplo, hemos tenido que utilizar la partícula `use` para poder ver la variable `$nombre`. Esto ocurre porque al ser una función **anónima**, hemos **perdido el ámbito (scope)**.*

Actividad de aprendizaje 2:

Crea un script donde haya una función de reciba 2 parámetros:

- *Una frase.*
- *Si se escribe en mayúsculas o minúsculas.*

Dentro de la función puedes utilizar los métodos `strtoupper()` y `strtolower()` para convertir el texto en mayúsculas o minúsculas en función de tus necesidades.

Llama a tu función con un texto y una indicación sobre si quieres que muestre tu texto en mayúsculas o minúsculas.

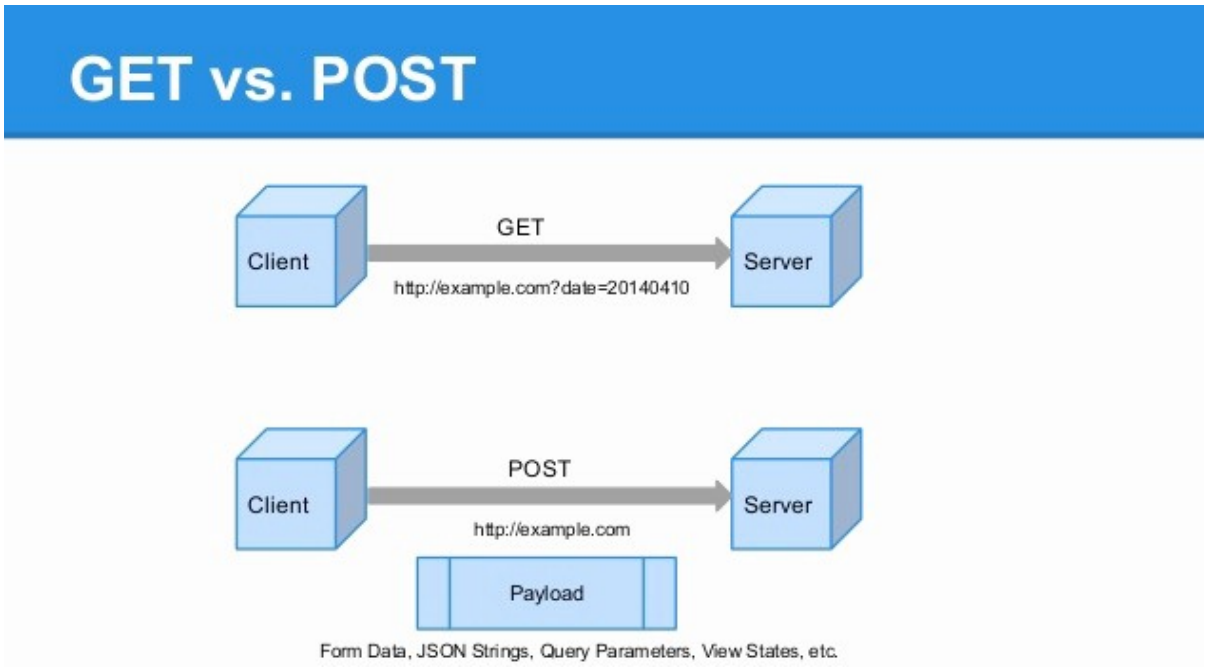
Sin compartir el código, comparte en el foro las estrategias que has utilizado para resolver este ejercicio.

8. RECUPERACIÓN Y UTILIZACIÓN DE INFORMACIÓN PROVENIENTE DEL CLIENTE WEB

El mecanismo de **comunicación entre un cliente y un servidor** se ubica en la página web. Si entendemos el cliente, ya no sólo como el navegador, sino como la persona que se sienta frente al ordenador, para que esta pueda interactuar con el servidor, debe enviarle **datos**.

Estos datos se mandan con alguna interfaz, habitualmente **formularios** como veremos más adelante, pero también caben otras tan dispares como APIs o asistentes de voz.

Sea cual sea la interfaz el cliente debe hacer la acción de "hablar" con el servidor. El verbo "hablar" lo podemos cambiar por los métodos de **comunicación GET y POST**.



Workflow de los métodos GET y POST. Fuente: https://mbilgil0.medium.com/. Esta imagen se reproduce acogiéndose al derecho de cita o reseña (art. 32 LPI), y está excluida de la licencia por defecto de estos materiales.

Veamos ahora las **diferencias** entre ambos métodos y cuando se suelen utilizar:

➔ **Diferencias**:

CARACTERÍSTICAS	GET	POST
Parámetros	Visible en la barra de direcciones para el usuario	Casi invisible para el usuario ya que los parámetros se colocan en el cuerpo
Caché y registro del servidor	Los parámetros URL se guardan sin cifrar.	Los parámetros URL no se guardan automáticamente
Comportamiento al actualizar el navegador o retroceder	Los parámetros URL no se envían de nuevo	El navegador advierte de que los datos del formulario se enviarán de nuevo
Restricciones de tipo de datos	Solo caracteres ASCII	Caracteres ASCII y datos binarios
Longitud de datos	Limitado al máximo del URL (2048 caracteres)	Ilimitado (depende del servidor)

➔ El **método GET** es útil cuando deliberadamente queremos que el cliente vea en la barra de direcciones que parámetros manda al servidor. Por ejemplo cuando buscamos algún producto en alguna tienda Online.

➔ El **método POST** se suele utilizar en formularios, con muchos parámetros o incluso cuando hay que adjuntar archivos.

9. INCLUSIÓN DE FICHEROS EXTERNOS. GESTIÓN DE FICHEROS Y DIRECTORIOS

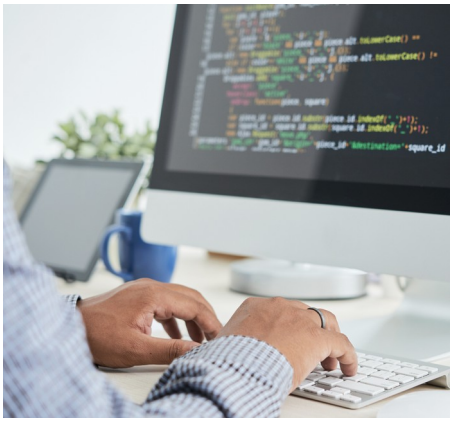
Los archivos en PHP se abren con la **función `fopen()`,** que requiere dos parámetros:

- El archivo que se quiere abrir.
- La forma en que abrir el archivo.

La función devuelve un **puntero** al archivo si es satisfactoria o **cero** si no lo es. Los archivos se abren para realizar operaciones de **lectura o escritura**. Veamos un ejemplo:

```
if (!$fp = fopen("archivo.txt", "r")){
    echo "No se ha podido abrir el archivo";
}
```

En el ejemplo hemos abierto el fichero en **modo lectura (r)**. Más tarde veremos el **modo escritura (w)**. Pero hay muchos más.



➔ **Conexión con el archivo**:

Los archivos pueden ser **locales**, como en el ejemplo anterior, o **direcciones remotas**, ya que `fopen()` lo que hace es crear una conexión que será necesaria cerrar posteriormente con `fclose()`.

Lo que hemos hecho hasta ahora ha sido crear una conexión con el archivo, pero todavía no hemos leído ni escrito nada en él.

➔ **Funciones de lectura**:

De funciones de lectura de archivos tenemos muchas: `readfile()`, `fgets()`, `fread()`, `fscanf()`, etc.

De momento el método más útil será `fgets()` ya que lee una longitud determinada, o hasta que encuentra un salto de línea, o hasta que encuentra el final del archivo (EOF). La sintaxis es: `string fgets (resource $handle [, int $length ])`

Un ejemplo de **lectura de un fichero**:

```
$fp = fopen("archivo.txt", "r");
while (!feof($fp)){
    $linea = fgets($fp);
    echo $linea;
}
fclose($fp);
```

➔ Función de escritura:

Para **escribir información** en archivos también tenemos varios métodos pero nos centraremos en `fwrite()`. Si quisiéramos escribir en un archivo haríamos algo como:

```
$fp = fopen("archivo.txt", "w");

fwrite($fp, "Hola qué tal");
fwrite($fp, ", cómo estás? \n");

fclose($fp);
```

En recursos para ampliar encontrarás tanto una lista completa con las funciones del sistema de archivos, como un listado de los modos posibles de `fopen()` usando mode.

Vídeo: trabajar con ficheros

Visualiza el siguiente vídeo en el que aprenderás a trabajar con ficheros.

10. INTERACCIÓN CON EL USUARIO: FORMULARIOS

*Los **formularios** son la interfaz más habitual para que un cliente mande información a un servidor. Esta transferencia de información, como ya hemos visto, se hace con los métodos GET y POST.*

Para probar el **método POST** necesitaremos crear un pequeño formulario. El formulario lo podemos realizar con **HTML** (cliente) y lo gestionaremos con **PHP** (servidor). Podríamos hacerlo todo con PHP, pero así ejemplificamos más qué hace cada uno de los actores.

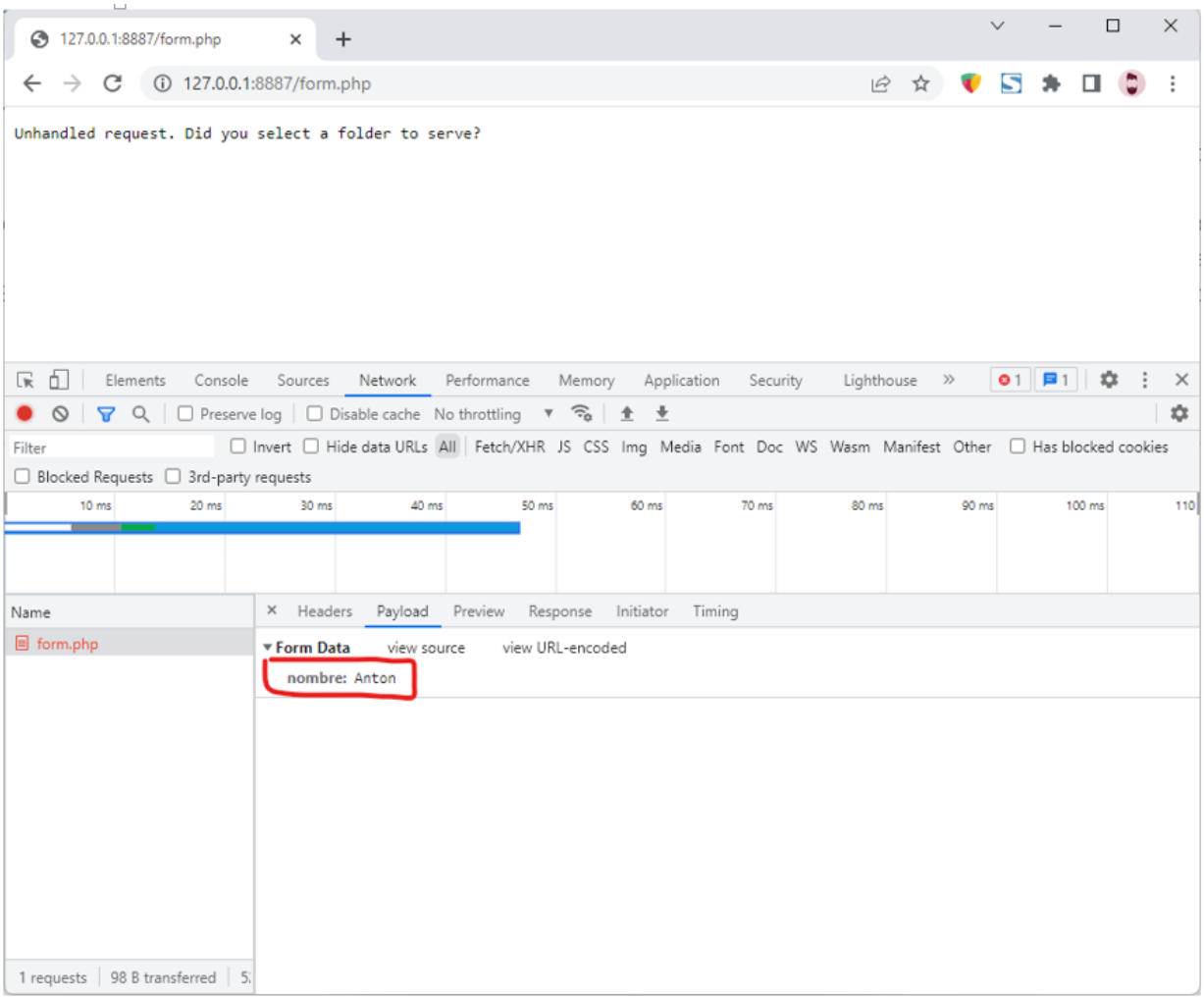
El siguiente **código** crea un formulario:

Código: creación de formulario

```
<html>
      <head>
            <title>Ejemplo de formulario</title>
      </head>
     <body>
            <form method="post" action="form.php">
            Nombre: <input type="text" name="nombre">
            <input type="submit">
            </form>
      </body>
</html>
```

- **I**. Este formulario tiene un sencillo campo de texto llamado "nombre" y un botón para mandar ese nombre al servidor.
- **II**. En el servidor deberemos alojar un script con el nombre **form.php** que recibirá el parámetro nombre gracias al método POST.

➔ **III**. Si ejecutamos ahora ese script y pulsamos el botón de enviar, vemos algo parecido a:



Resultado del paso de parámetros.

➔ **IV**. ¿Por qué no funciona? Pues, aunque vemos que el formulario ha mandado correctamente el nombre (Anton) hacia el servidor con el método POST, aún no tenemos creado el script **form.php**. Esto lo veremos en el siguiente apartado.

11. PROCESAMIENTO DE LA INFORMACIÓN INTRODUCIDA EN UN FORMULARIO. MÉTODOS POST Y GET. PASO DE VARIABLES A TRAVÉS DE FORMULARIO Y DE LA URL

En el apartado anterior hemos creado un formulario, pero nos faltaba el *script* que recogiera los datos enviados e hiciera algo. Vamos a crear un *script* llamado **form.php**:

Código: creación de script form.php

```
<?php
	$nombre = $_POST['nombre'];
	if (empty($nombre))
	{
		echo "Hola desconocido";
	}
	else
	{
		echo "Hola $nombre";
	}
?>
```

¿Qué hace este ***script***? Veámoslo **paso a paso**:

➔ **Paso 1**.

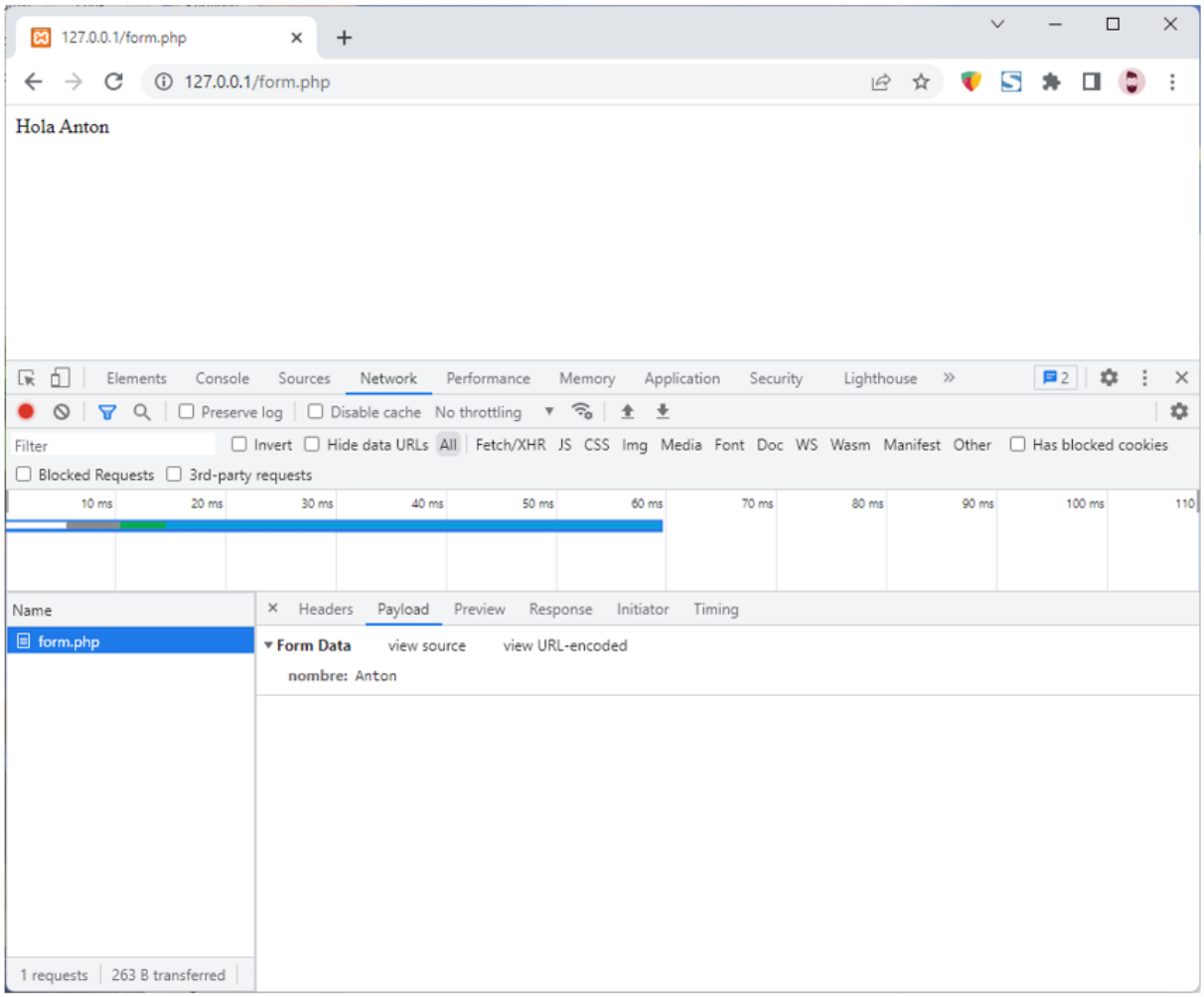
Recibe el nombre con `$_POST['nombre']` y lo guarda en una variable llamada también `$nombre`.

- ✔ Comprueba si nombre existe, es decir, si hay algo guardado dentro.
- ✔ Si encuentra algo, saluda con `echo "Hola $nombre";`.

✔ Si no encuentra un nombre, saluda con `echo "Hola desconocido";`.

➔ **Paso 2**.

Si ahora ejecutamos el *script*, veremos:



Resultado del paso de parámetros

➔ **Paso 3**.

El paso de parámetros también se puede hacer con GET. Para ello no hace falta montar un formulario. En el lado del cliente ponemos un sencillo enlace como: `<a href="form.php?nombre=Anton">Click</a>`.

Y en el servidor, en **form.php**, cambiamos `$_POST['nombre'];` por `$_GET['nombre'];` y conseguimos el mismo resultado.

Encontrarás un vídeo explicativo de cómo enviar y recibir datos con GET y POST en recursos para ampliar.

Actividad de aprendizaje 3: formularios

La actividad se elaborará en parejas.

Debemos crear un formulario donde escribiendo los datos de una persona se calculará su IMC (índice de masa corporal) a partir de la siguiente fórmula:

$$IMC = peso / estatura2$$

Un alumno deberá crear el formulario y otro el script del servidor. Deberéis pactar los nombres de las variables a pasar, así como el método utilizado (GET o POST).

Al final el cliente debe ver su IMC y la frase que le corresponda según la siguiente tabla:

MC	Nivel de peso
Por debajo de 18.5	Bajo peso
18.5 – 24.9	Normal
25.0 – 29.9	Sobrepeso
30.0 o más	Obesidad

Utilizad comentarios en el código para saber qué hace cada bloque que hayáis diseñado.

Cuestionario

Lee el enunciado e indica la opción correcta:

Los tipos de datos compuestos en PHP son:

a. Null, resource.

b. Array, object, callable, iterable.

c. Boolean, integer, float, string.

Lee el enunciado e indica la opción correcta:

Los parámetros visibles en la barra de direcciones corresponden al método:

a. COOKIE.

b. POST.

c. GET.

Lee el enunciado e indica la opción correcta:

La partícula "use" sirve:

a. Para poder utilizar arrays.

b. Para utilizar variables fuera del scope de una función anónima.

c. La partícula "use" no es propia del lenguaje PHP.

RESUMEN

En esta unidad ya hemos entrado de lleno en el **desarrollo de aplicaciones** del lado del **servidor con PHP**.

Así como en la unidad 2 creamos un embrión de aplicación web ahora ya deberemos ser capaces de **crear aplicaciones que puedan tomar decisiones**, reaprovechar código mediante funciones, iterar mediante bucles, etc.

A parte de aprender la **sintaxis del lenguaje** y sus estructuras básicas, también hemos aprendido a mandar datos del cliente hacia el servidor y procesar esos datos gracias al uso de formularios y a los mecanismos de comunicación GET y POST.

Con lo que sabemos ya podemos crear programas más o menos complejos. En futuras unidades aprenderás técnicas más avanzadas sobre el **desarrollo web**.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- Funciones del sistema de archivos: *https://www.php.net/manual/es/ref.filesystem.php* [Consulta noviembre 2022].
- Lista de los modos posibles de `fopen()` usando mode: *https://www.php.net/manual/es/function.fopen.php* [Consulta noviembre 2022].
- Listado de operadores en PHP: *https://www.w3schools.com/php/php_operators.asp* [Consulta noviembre 2022].
- Vídeo: Enviar y recibir datos con GET y POST: *https://www.youtube.com/watch?v=ZKJESVm5otA* [Consulta noviembre 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- Material y apuntes para certificación de PHP: *https://diego.com.es/certificacion-php* [Consulta noviembre 2022].
- Referencia no oficial de PHP: *https://www.w3schools.com/php/default.asp* [Consulta noviembre 2022].
- Referencia oficial de PHP: *https://www.php.net/* [Consulta noviembre 2022].

GLOSARIO

- **Array**: estructura de datos que permite almacenar varios valores en una sola variable.
- **Bucles**: los bucles se utilizan para ejecutar el mismo bloque de código una y otra vez, siempre que se cumpla una determinada condición.
- **Función**: sección de un programa que calcula un valor de manera independiente al resto del programa.
- **Operadores**: los operadores se utilizan para realizar operaciones en variables y valores.
- **Variable**: las variables son "contenedores" para almacenar información.