PHP.

Concepto.

- Hipertext Preprocessor – Preprocesador de Hipertexto.
- Lenguaje de alto nivel interpretado que se inserta dentro de una página web y es ejecutado en un servidor.
- Puede compararse a otros lenguajes de script similares como JSP (Java Server Pages) y ASP (Active Server Pages).
- El resultado de la ejecución del código PHP en el servidor se integra en la página web que luego es enviada al navegador, que ignora el procesamiento realizado en el servidor.
- La finalidad es realizar páginas web dinámicas cuyo contenido puede ser generado completa o parcialmente al llamarse a la página, gracias a la información contenida en un formulario u obtenida de una base de datos.

Historia.

- https://es.wikipedia.org/wiki/PHP#Visi%C3%B3n_general

Versiones.

- https://es.wikipedia.org/wiki/PHP#Historial_de_lanzamiento

Funcionamiento.

- Un navegador solicita una página web a un servidor (petición HTTP con página PHP), en este caso con extensión PHP (pagina.php, por ejemplo. Para que un archivo sea interpretado como PHP, debe tener la extensión PHP, no HTML.).
- El servidor web (Apache, IIS, …), recibe la petición y solicita al interprete PHP (otro programa que se ejecuta en el servidor), que lea el archivo PHP.
- El intérprete PHP ejecuta los comandos contenidos en el archivo (página PHP), pudiéndose comunicar con otros programas, como gestores de bases de datos (MySQL, MariaDB, SQL server, Oracle Database, …).
- Tras ejecutar el programa (código PHP), contendido en el archivo (página PHP), envía éste al servidor.
- El servidor web envía la página (HTML) al cliente que la había solicitado, y ésta es mostrada en el navegador (repuesta HTTP).
- La salida de los resultados en PHP es la página web (HTML).
- Una página PHP puede contener HTML, CSS y JavaScript.

Ciclo de vida de una aplicación web.

- El flujo típico de una aplicación web es el siguiente:
    - El cliente hace una petición al servidor (envía un comando HTTP).
    - El servidor recibe la petición y la analiza:
        - Si se trata de código PHP se lo pasa al intérprete PHP.

        - El intérprete procesa el código y genera una salida (normalmente HTML).
    - El servidor envía dicho resultado al cliente.
    - El cliente analiza el envío y genera la página web.
    - Si los hay, se ejecutan scripts del lado cliente (JavaScript).

Herramientas de desarrollo y servidores locales para probar PHP y MySQL.

- Permiten aprender y probar nuestro trabajo (páginas web, programas, etc.), a nivel local sin necesidad de acceso a Internet.
- Permiten replicar un servidor en producción en una instalación local para experimentar sin miedo a afectar al sistema real.

WAMPSERVER

- Sólo versión para Windows.
- Descarga:
    - http://www.wampserver.es/
    - https://www.wampserver.com/en/

WAMP/LAMP/MAMP.

- Paquete de software libre Windows | Linux | Multiplataforma Apache MySQL PHP.
- Descarga:
    - https://bitnami.com/stacks/infrastructure

EASYPHP.

- Sólo para Windows.
- Descarga:
    - https://www.easyphp.org/

XAMPP.

- Herramienta de desarrollo que permite probar nuestro trabajo (páginas web, programas, etc.), a nivel local sin necesidad de acceso a Internet.
- Descarga:
    - https://www.apachefriends.org/es/index.html
- Incluye:
    - Servidor web **Apache**.
    - SGBD **MariaDB** que sustituye a MySQL.
    - phpMyAdmin para gestionar bases de datos.
    - **PHP**:
        - Preprocesador de hipertexto - Hypertext PreProcessor.
        - Lenguaje de scripts para desarrollo web del lado servidor.
        - El código PHP se ejecuta en el servidor.
    - Servidores **FTP** como Filezilla Server.

- **Webalizer**.
    - Software de analítica web.
- Otras aplicaciones:
    - **Apache Tomcat, OpenSSL, XML, PERL**.

LOCALHOST.

- Dispositivo local, servidor local o anfitrión local.
- Se refiere al ordenador o dispositivo local que se está usando, es decir, al propio equipo.
- **Acceso al servidor local**:
    - Con nombre:
        - localhost
    - Con dirección IP:
        - IPv4 → 127.0.0.1
        - IPv6 →::1 o 0000:0000:0000:0000:0000:0000:0000:0001
- Rutas de acceso al sitio web:
    - https://localhost/ruta de acceso/nombre de archivo.php
    - Ejemplos:
        - https://localhost/sitios/archivo1.php
        - https://127.0.0.1/sitios/archivo1.php
        - https://[::1]/sitios/archivo1.php
        - https://localhost/sitios/prueba/archivo1.php
        - https://localhost/archivo1.php
- **Dashboard**.
    - Si sólo se escribe localhost en el navegador se accede al panel de control o dashboard de XAMPP.
- **Carpeta htdocs**.
    - Carpeta que incluye XAMPP y que sirve para el almacenamiento de los archivos HTML, PHP, sitios web en general, etc., a los que posteriormente se accederá mediante un navegador.
- **Abrir un archivo PHP en el navegador.**
    - Abrir XAMPP e iniciar el servidor Apache. Si está detenido, no se pondrá procesar el código PHP.
    - El archivo con el código PHP se debe guardar en la carpeta “htdocs” de XAMPP o en alguna subcarpeta de ella creada para organizar mejor los contenidos.
    - Abrir el navegador.
    - En la barra de direcciones, escribir la ruta de acceso al archivo PHP guardado en “htdocs”, usando el siguiente formato ya visto.
        - https://localhost/ruta de acceso/nombre de archivo.php

PHP y HTML.

- Una página web puede tener una o varias inserciones PHP.

- El código PHP genera texto que se integra en la página HTML enviada al navegador.
- **Tipo de texto generado**:
    - Texto plano.
    - Código HTML.
    - Código JavaScript.
    - Otros (XML, JSON, …).
- **Inserción**:
    - Fuera de las etiquetas HTML y antes de la instrucción <!doctype html>.
    - En la cabecera <head>.
    - En el cuerpo <body>.
    - En varias a la vez.
- **Incluir un archivo PHP dentro de otro.**
    - En un archivo con extensión HTML no se puede incluir un archivo PHP, hay que cambiar su extensión a PHP.
    - Para incluir un archivo PHP dentro de otro se usan las instrucciones:
        - Pueden usarse sin paréntesis.
        - include()
            - Permite incluir un archivo en otro.
            - Si se produce un error y no se encuentra el archivo, PHP emite una advertencia (warning), y se continúa ejecutando el script.
        - require().
            - Funciona igual que include() pero si se produce un error y no se encuentra el archivo, PHP emite un error fatal y el script se detiene.
            - Uso en aplicaciones que necesiten obligatoriamente algún archivo crítico para continuar, porque sin él, no pueden.
        - Sintaxis:
            - include(“ruta de acceso/archivo.php”);
            - require(“ruta de acceso/archivo.php”);
        - Ejemplos.
            - include(“archivo1.php”);
            - require(“misitio/PHP/archivo2.php”);

SINTAXIS BÁSICA DE PHP.

Marcas o etiquetas PHP

- **<?php…..?>**
    - Etiqueta que delimita el código PHP dentro de la página web.
    - Lo que haya ente las 2 etiqueta se procesará por el intérprete PHP
    - Puede haber varios fragmentos de código a lo largo de una página web delimitado por estas etiquetas.
    - Para que el código sea ejecutado, la página web que lo contenga debe guardarse con la extensión PHP (página dinámica), y no HTML (página estática).

    - Se puede incluir código PHP tanto en la cabecera <head>, como en el cuerpo <body> de la página web.
- **Comentarios.**
    - Son ignorados y sirven para documentar el código, facilitar su comprensión, añadir ideas, sugerencias, etc., …
    - Sintaxis:
        - De una línea:
            - // Texto del Comentario.
            - # Texto del Comentario.
        - De varias líneas:
            - /* Texto del Comentario Línea 1
            - Texto del Comentario Línea 2
            - Texto del Comentario Línea N */
- **Fin de instrucción o separador de instrucciones**.
    - Cada instrucción distinta se separa con un punto y coma (;).

TIPOS DE DATOS.

- PHP admite 10 tipos de datos primitivos.

Escalares

- Solo pueden contener un valor.
- Tipos:
    - Números enteros, integer o int.
    - Numero reales, de coma flotante o float (antiguamente double).
    - Cadenas de caracteres o string
    - Booleanos o boolean.

Compuestos.

- Pueden contener varios valores.
- Tipos:
    - Arrays.
    - Objetos.
    - Iterables.
    - Callback o Callables.

Especiales.

- Resource (recurso).
- Null.

TIPOS ESCALARES.

Tipos de datos primitivos.

- Tipos de datos elementales que proporcionan los lenguajes de programación.

- Un tipo primitivo está predefinido en un lenguaje y puede ser manipulado mediante operaciones que también incorpora el lenguaje de programación.
- Son valores simples que se pueden agrupar para formar tipos estructurados.

Enteros (integer o int).

- Numero enteros sin decimales.
- Los valores pueden ser positivos, negativos o cero.
- Formatos de almacenamiento.
- **Base 10.**
  - Sistema numérico decimal.
  - Ejemplo:
    - $numero = 200.
- **Base 8 (octal).**
  - Se antepone un 0 (cero), al número.
  - Ejemplo:
    - $numero = 0200.
- **Base 16 (hexadecimal).**
  - Se antepone 0x al número.
  - Ejemplo:
    - $numero = 0x200.
- Aunque se asigne un valor en octal o hexadecimal, se muestra luego en decimal (por ejemplo, con echo).
- **Números negativos**.
  - Se incluye un guion delante del número.
  - Ejemplo:
    - $numero = -200.

Reales o de coma flotante (float o double).

- Numero enteros con decimales.
- Los valores pueden ser positivos, negativos o cero.
- Una suma ente un entero y un real da como resultado un número real.
- Ejemplo:
  - $numero1 = 0.80;
  - $numero2 = 127.57;
  - $numero3 = -80.90;

Cadenas de caracteres.

- Conjunto de caracteres alfanuméricos entre comillas dobles o simples.
- Las comillas dobles permiten concatenar texto y variables sin usar el operador punto (.).
- Las comillas simples muestran el nombre literal de una variable y no su contenido si no se concatena con el operador punto.
- Los caracteres de escape \" o \' permiten incluir comillas literales como un carácter más.
- Uso comillas dobles:

    - Permite incluir variables para imprimirlas literalmente junto con texto.
    - Ejemplos:
        - $texto1 = “Esto es una cadena de texto”;
        - $vacia1 = “”;
- Uso comillas simples:
    - Permiten imprimir el contenido íntegro de los caracteres entre comillas.
    - Ejemplos:
        - $texto2 = “Esto es una cadena de texto”;
        - $vacia2 = “”;
- Ejemplo uso de las distintas comillas.
    - $nombre = “Rosa”
    - $nombre1 = “Te llamas: $nombre”;
    - $nombre2 = ‘Te llamas: $nombre’;
    - echo $nombre1; // Resultado: Te llamas Rosa.
    - echo $nombre2; // Resultado: Te llamas $nombre;

Valores booleanos (boolean).

- Solo pueden contener uno de 2 valores: true o false.
- Si se muestra el contenido de una variable por pantalla con **echo**:
    - Si tiene el valor ***true*** muestra 1.
    - Si tiene el valor ***false*** no muestra nada.
- Se pueden convertir otros datos a booleano según las siguientes reglas.
- Conversión a false:
    - El cero entero (0).
    - El cero con decimales (0.000…).
    - Cadenas vacías (“”).
    - Cadenas igual a cero (“0”).
    - Tablas vacías.
    - Objetos vacíos
    - null.
- Conversión a true:
    - Valores enteros positivos o negativos.
    - Valores reales con decimales positivos o negativos.
    - Cadenas de caracteres.

TIPOS COMPUESTOS.

Arrays.

- Estructura de datos que permite almacenar varios valores.
- A los valores se accede por su posición en el array usando un índice.
- Los datos almacenados en un array pueden ser de cualquier tipo.

Objetos.

- Instancias de una clase.
- Una clase define los datos y la lógica de un objeto.

- La lógica se divide en funciones (métodos) y variables (propiedades).

Iterables.

- En un pseudotipo.
- Cualquier valor que se puede recorrer con un bucle foreach().
- Acepta cualquier array u objeto que implemente la interfaz Traversable.
- Traversable es un interfaz que permite detectar si una clase puede recorrerse mediante foreach.
- Interfaz:
    - Clase que sólo contiene constantes y declaraciones de métodos sin implementar o desarrollar.
    - Se usan para indicar qué métodos debe obligatoriamente implementar una clase.
    - Los métodos de una interfaz son siempre públicos.

Callback o Callables.

- Funciones de llamada, llamadas de retorno, o retro llamadas.
- Función que se pasa a otra función como argumento.
- Pueden utilizarse también como métodos de objeto.

TIPOS ESPECIALES.

Recurso o Resource.

- Tipo de dato que contiene una referencia a un recurso externo como un archivo abierto, una conexión de base de datos, etc., …
- Esta referencia luego podrá será utilizada por las funciones nativas de PHP.

Nulo o Null.

- Una variable sin valor por no haber sido inicializada antes posee el valor nulo, es decir, no contiene nada.
- Valores válidos null, NULL, Null, …

CARACTERES O SECUENCIAS DE ESCAPE.

\n -> Salto de línea.

\r -> Retorno de carro.

\t -> Salto de tabulación.

\v -> Tabulación vertical.

\e -> Escape.

\” -> Comillas dobles.

\f -> Página siguiente.

**** ->Barra inclinada inversa.

\$ -> signo dólar.

\nnn -> Carácter ASCII nnn en octal.

\xnn -> Carácter ASCII nn en hexadecimal.

\u{nnnnnn} -> Carácter Unicode nnnnnn en hexadecimal.

VARIABLES.

Concepto.

- Espacio de almacenamiento reservado en memoria.
- Queda definida por una posición y un nombre.

Nombre de una variable.

- Empiezan por $:
  - $edad, $nombre, $precio.
- Distinción entre mayúsculas y minúsculas:
  - $numero ≠ $NUMERO ≠ $Numero
- Caracteres no permitidos:
  - #, $, % o &
- Puede empezar por letra o guion bajo (_).
- No pueden empezar por números, pero si contenerlos.
- Uso de nomenclatura Camel Case para nombres con varias palabras:
  - $NombreDelCliente

Definición.

- No hay instrucción de creación.
- Se crea la variable y se le asigna un valor.
- Una nueva asignación cambia el dato o el tipo de la variable, ya que PHP es un lenguaje débilmente tipado.
- Sintaxis:
  - $nombre de la variable = valor;
- Ejemplos:
  - $nombre = “Eugenia”; // Cadena de caracteres.
  - $edad = 30; // Número entero.
  - $sueldo = 40.25; // Número real o de coma flotante.
  - $prestado = true; // Booleano.
  - $edad = 40; // Nuevo valor.
  - $prestado = 1000.50; // Nuevo valor y nuevo tipo.

Variables variables o Variables dinámicas.

- Variable que contiene el nombre de otra variable.
- La variable en nombrada con el valor almacenado en otra variable.
- Sintaxis:
    - $$variable = valor;
    - El primer $ representa a la variable exterior.
    - $variable representa a la variable interior.
    - La variable interior es sustituida por el valor de la variable que se usa como nombre de la variable por el $ exterior.
    - Para acceder a el valor de una variable variable hay que encerrarla ente llaves {}.
- Ejemplos:
    - $Toledo = 200000;
    - $Teruel = 50000;
    - $Nombre_Ciudad = “Toledo”;
    - $Datos_Toledo = $$Nombre_Ciudad;
    - echo “Número de habitantes de $Nombre_Ciudad es ${$Nombre_Ciudad}.”;
        - Salida: Número de habitantes de Toledo es 200000.
    - echo “Número de habitantes de $Nombre_Ciudad es $Datos_Toledo”;
        - Salida: Número de habitantes de Toledo es 200000.
    - $Nombre_Ciudad = “Teruel”;
    - echo “Número de habitantes de $Nombre_Ciudad es ${$Nombre_Ciudad}”;
        - Salida: Número de habitantes de Teruel es 50000.

Ámbito y duración de una variable.

- El alcance de una variable es el script donde está definida, no puede usarse en otro script (o página).
- La duración de una variable es el tiempo de ejecución del script. Cuando el script termina, se eliminan las variables.
- Sin embargo, se puede conservar el valor de una variable más allá de la ejecución del script o pasar su valor a una variable de otro script.
- **Tipos**:
    - **Variable local.**
        - Variable definida dentro de una función.
    - **Variable global.**
        - Variable definida fuera de una función, o fuera de un método o de los parámetros de una clase.
        - Una variable global no está disponible dentro de una función.
        - **global:**
            - Instrucción que permite que se pueda usar una variable global dentro de un ámbito local.
            - Sintaxis:
                - global $Nombre de la variable = valor;
            - Ejemplo:
                - global $saldo = 3000;

- **Variable estática**.
    - Variable que sólo existe en el ámbito local de una función, pero no pierde su valor cuando termina la ejecución de ésta y el script continua, es decir, persiste tras la ejecución de la función que las contiene.
- **Variable superglobal**.
    - Son variables predefinidas en PHP que se pueden usar en cualquier punto de un programa, independientemente del alcance que tenga una línea de código.
    - Está disponible en cualquier parte de un script.
    - No es necesario emplear **global $variable.**
    - Se pueden usar desde cualquier función, clase o archivo PHP.
    - No pueden ser usadas como variables variables dentro de funciones o métodos de clase.
    - Las variables superglobales son:
        - *$GLOBALS*
        - *$_SERVER*
        - *$_GET*
        - *$_POST*
        - *$_FILES*
        - *$_COOKIE*
        - *$_SESSION*
        - *$_REQUEST*
        - *$_ENV*

FUNCIONES PARA VARIABLES.

Funciones alias.

- Funciones que se puede llamar con más de un nombre.
- Al actualizarse PHP se han creado nuevas funciones que hacen lo mismo que otras antiguas.
- Algunas de las antiguas se mantienen por compatibilidad con versiones anteriores de PHP, pero pueden desaparecer en un futuro.

isset().

- Indica si una variable ha sido declarada y su valor es distinto de *null*.
- Devuelve un valor booleano (true-false).
- Uso frecuente con instrucciones condicionales.
- Sintaxis:
    - isset(nombre de la variable)
- Ejemplo:

```
if(isset($edad)
{
        echo “La variable existe”;
}
```

```
else
{
        echo “La variable no existe”;
}
```

empty().

- Indica si una variable está vacía o no.
- Una variable esta vacía si no existe (!isset()), o su valor es igual a *false*. ($variable == false).
- Devuelve un valor booleano:
    - **false**.
        - Devuelve *false* si tiene un valor no vacío distinto de cero.
    - **true.**
        - Devuelve *true* con los siguientes valores, que se consideran vacíos:
            - Cadena vacía (“”), 0, 0.0, “0”, ”0,0”, false, null, arrays vacíos.
- Uso frecuente con instrucciones condicionales.
- Sintaxis:
    - empty(nombre de la variable)
- Ejemplo:

```
if(empty($edad)
{
        echo “La variable está vacía”;
}
else
{
        echo “La variable no está vacía”;
}
```

is_null().

- Comprueba si el valor de una variable es *null*.
- Devuelve un valor booleano (true-false).
- Uso frecuente con instrucciones condicionales.
- Sintaxis:
    - is_null(nombre de la variable)
- Ejemplo:

```
if(is_null($edad)
{
        echo “El valor de la variable es null”;
}
else
{
        echo “El valor de la variable no es null”;
```

```
}
```

unset().

- Elimina una variable u objeto de la memoria.
- Tras eliminar una variable con unset(), al preguntar por ella con isset(), esta función devolverá ***false***.
- Sintaxis:
  - unset(nombre de la variable u objeto)
- Ejemplo:

```
unset($edad);
```

is_integer().

- Devuelve un valor booleano (true-false), si una variable es de tipo entero.
- Uso frecuente con instrucciones condicionales.
- Alias de is_int().
- Sintaxis:
  - is_integer(nombre de la variable)
- Ejemplo:

```
$numero = 7;
if_integer($numero)
{
        echo “El número es un entero”;
}
else
{
        echo “El número no es un entero”;
}
```

is_double().

- Devuelve un valor booleano (true-false), si una variable es de tipo real.
- Uso frecuente con instrucciones condicionales.
- Alias de is_float().
- Sintaxis:
  - is_double(nombre de la variable)
- Ejemplo:

```
$numero = 20.45;
if_double($numero)
{
        echo “Es un número con decimales”;
}
else
{
```

```
        echo “No es un número con decimales”;
}
```

is_string().

- Devuelve un valor booleano (true-false), si la variable es de tipo texto.
- Uso frecuente con instrucciones condicionales.
- Sintaxis:
  - is_string(nombre de la variable)
- Ejemplo:

```
$texto = “PHP”;
if_string($texto)
{
        echo “Es un texto”;
}
else
{
        echo “No es un texto”;
}
```

is_bool().

- Devuelve ***true*** si una variable es de tipo booleana y ***false*** si no lo es.
- Uso frecuente con instrucciones condicionales.
- Sintaxis:
  - is_bool(nombre de la variable)
- Ejemplo:

```
$verdad = true;
if_double($verdad)
{
        echo “Si, si es booleana”;
}
else
{
        echo “No, no es booleana”;
}
```

intval().

- Convierte el valor de una variable a entero.
- Sintaxis:
  - intval(nombre de la variable)
- Ejemplo:
  - $texto = “40”;
  - intval($texto);
  - is_integer($texto) devolverá *false*.

doubleval().

- Convierte el valor de una variable a un número real.
- Sintaxis:
  - doubleval(nombre de la variable)
- Ejemplo:
  - $texto = “30.5”;
  - doubleval ($texto);
  - is_double($texto) devolverá *false*.

strval().

- Convierte el valor de una variable a una cadena de caracteres.
- Sintaxis:
  - strval(nombre de la variable)
- Ejemplo:
  - $numero = 40;
  - strval($numero)
  - is_string($numero) devolverá *false*.

settype().

- Convierte una variable de un tipo a otro tipo.
- Sintaxis:
  - settype(nombre de la variable a convertir, “nuevo tipo”);
  - Tipos:
    - "boolean" o "bool”.
    - "integer" o "int".
    - "float" o "double".
    - "string".
    - "array".
    - "object".
    - "null".
- Ejemplo:
  - $numero = 40;
  - $texto = “20”;
  - settype($numero, “float”);
  - settype($texto, “int”);

gettype().

- Devuelve el tipo de una variable.
- Tipos devueltos:
  - “boolean".
  - "integer".
  - "double".
  - "string".
  - "array".

    - "object".
    - "resource".
    - "NULL".
    - "unknown type".
- Sintaxis:
    - gettype(nombre de la variable);
- Ejemplo:
    - $numero = 40;
    - $texto = “Hola”;
    - gettype($numero); Devuelve integer.
    - gettype($texto); Devuelve string.

CONVERSIÓN DE TIPOS.

- **Conversión implícita.**
    - Existe una conversión implícita que desarrolla PHP según se necesite.
    - Un valor puede ser convertido de forma automática a uno de otro tipo cuando un operador, función o estructura de control lo requiera.
    - Ejemplo:

      $numero1 = 1.5; Variable float.
      $numero2 = 2; Variable entera.
      $suma = $numero1 * $numero2; Variable float.
      El valor de $numero2 se ha convertido a float para realizar la operación.
      echo gettype($suma); Devuelve "double".

- **Conversión explicita o forzado de tipos.**
    - Se fuerza la conversión de uno a otro tipo con:
        - Funciones como:
            - intval(), doubleval(), stringval(), is_bool(), settype() y otras.
        - Casting:
            - Se escribe el nombre del tipo entre paréntesis delante de la variable a forzar.
            - Forzados permitidos:
                - (int), (integer).
                    - Forzado a integer.
                - (bool), (boolean).
                    - Forzado a boolean.
                - (float), (double), (real).
                    - Forzado a float.
                - (string).
                    - Forzado a string.
                - (array).
                    - Forzado a array.
                - (object).
                    - Forzado a object.

        - (unset).
            - Forzado a NULL.
            - No elimina la variable ni tampoco su valor.
        - (binary).
    - Sintaxis:
        - (tipo a forzar) $nombre de la variable;
    - Ejemplo**:**
        - $texto = “20”**;**
        - $numero = (int) $texto;
        - echo gettype($numero); // Mostrará como resultado integer.

CONSTANTES.

- **Concepto.**
    - Tipo de dato cuyo valor no puede variar durante la ejecución de un script o programa, a excepción de las constantes mágicas.
    - Pueden contener cualquier tipo de dato escalar.
    - El acceso a una constante es global. Se puede acceder a ellas desde cualquier sitio del script.
    - Para concatenar constantes hay que usar el operador punto, no sirve unirlas entre ellas, o a una cadena, encernadándolas entre comillas dobles.
- **Alcance y duración de una constante**.
    - El script donde está definida.
    - Duración: mientras se ejecuta el script.
- **Creación**.
    - Nombre:
        - El nombre no puede empezar por el carácter $ porque si no, se crearía una variable.
        - Letras, números o guion bajo.
        - No pueden empezar por número.
        - Es costumbre escribir el nombre en mayúsculas.
    - Sintaxis:
        - define (“NOMBRE DE LA CONSTANTE”, valor);
    - Ejemplos:
        - define (“IVA”, 21)**;**
        - define (“PI”, 3.1416);
        - define (“MONEDA”, “euros”);
        - define (“ESTADO”, false);
- **Constante tipo array.**
    - Constante con más de un valor.
    - Sintaxis:
        - Creación:
            - define (“NOMBRE DE LA CONTANTE”, [valor1, valor2,…,valor n]);
        - Acceso a valores:

        - NOMBRE DE LA CONSTANTE[índice];
    - Ejemplo:
        - define (“COLORES”, [rojo, verde, azul]);
        - echo COLORES[1]; //Muestra verde.
- **Constantes de clase.**
    - Constantes que se definen dentro de clases.
    - La visibilidad predeterminada de las constantes de clase es *public*.
    - Los interfaces también pueden tener constantes, estás se comportan como constantes de clase.
    - El valor de una constante de clase se asigna una vez al declararla dentro de la clase, y no cada vez que se crea un objeto o una instancia de la clase.
    - Sintaxis:
        - const NOMBRE DE LA CONSTANTE = valor;
    - Ejemplo:

```
class miClase
{
        const A = 10.50;
        public const B= “PHP”;
        private const C=20;
        public miMetodo()
        {
                echo $this::B;
        }
}
```

- **Constantes predefinidas.**
    - Incluidas por defecto en PHP.
    - Se escriben en mayúsculas.
    - Función informativa:
        - Muestran información del servidor, versión de PHP, información de depuración, tamaño números, etc.
    - Sintaxis:
        - echo NOMBRE DE LA CONSTANTE;
        - echo “texto”. NOMBRE DE LA CONSTANTE;
    - **Algunas constantes:**
        - **PHP_SAPI.**
            - Muestra información sobre la API del servidor.
        - **PHP_VERSION.**
            - Muestra la versión de PHP ejecutándose en el servidor.
        - **PHP_OS.**
            - Muestra el sistema operativo del servidor.
        - **PHP_EXTENSION_DIR.**
            - Ruta de instalación de las extensiones de PHP.
        - **PEAR_INSTALL_DIR.**
            - Ruta de instalación de PEAR.
            - PEAR:
                - PHP Extension and Application Repository.

        - Conjunto de bibliotecas de código PHP que permiten hacer ciertas tareas de manera más rápida y eficiente reutilizando código escrito previamente por otras personas.
    - **PEAR_EXTENSION_DIR.**
        - Ruta de instalación de las extensiones de PEAR.
    - **PHP_LIBDIR.**
        - Ruta de almacenamiento de las librerías.
    - **PHP_EOL.**
        - Muestra el símbolo 'Fin De Línea' correcto de la plataforma en uso.
    - **PHP_INT_SIZE**.
        - El tamaño de un número entero en bytes en esta compilación de PHP.
    - **PHP_INT_MAX.**
        - El número entero más grande admitido en esta compilación de PHP.
    - **PHP_INT_MIN.**
        - El número entero más pequeño admitido en esta compilación de PHP.
    - **PHP_FLOAT_MIN.**
        - El menor número positivo de punto flotante representable.
        - Para el menor negativo se pone un guion delante de la constante.
    - **PHP_FLOAT_MAX.**
        - El mayor número positivo de punto flotante representable.
        - Para el mayor negativo se pone un guion delante de la constante.
    - **PHP_FLOAT_DIG.**
        - Número de dígitos decimales que se pueden redondear en un float y revertirlos si pérdida de precisión.
- **Constantes mágicas.** (En realidad no son constantes)
    - Son también constantes predefinidas.
    - Se denominan mágicas porque cambian dependiendo de donde se usan.
    - Se distinguen por su forma de escribirlas:
        - __NOMBRE DE LA CONSTANTE__
    - **Algunas constantes:**
        - **__LINE__**
            - El número de línea desde la que se imprime.
        - **__FILE__**
            - Ruta al fichero que la invoca.
        - **__DIR__**
            - Directorio del archivo.
        - **__FUNCTION__**
            - Muestra el nombre de la función actual.
        - **__CLASS__**
            - Nombre de la clase.

        - **__TRAIT__**
            - El nombre del rasgo.
            - Rasgo.
                - Mecanismo de reutilización de código en lenguajes de herencia simple como PHP.
                - Permiten reutilizar grupos de métodos sobre varias clases independientes que no están relacionadas jerárquicamente.
        - **__METHOD__**
            - Muestra el nombre de la función actual y el de la clase donde está incluida.
        - **__NAMESPACE__**
            - El nombre del espacio de nombres actual.
- **Funciones para constantes.**
    - **defined().**
        - Función que comprueba si una constante está o no definida.
        - Devuelve ***true*** o ***false***.
        - Sintaxis:
            - defined("NOMBRE DE LA CONSTANTE")
        - Ejemplo:

```
define("PI",3.1416);
if(defined("PI"))
{
        echo "El valor de PI es" . PI;
}
else
{
        echo "La constante PI no existe";
}
```

SALIDA DE DATOS.

echo().

- Función que permite mostrar cadenas de caracteres, contenido de variables, constantes, Arrays, etc. en el documento web, es decir, imprime por pantalla un contenido.
- No son necesarios los paréntesis.
- Sintaxis:
    - echo ("Texto");
    - echo "texto";
    - echo ($variable);
    - echo ("texto $variable");
    - echo ("texto" . $variable);
    - echo ('texto' . $variable);
- Ejemplos:
    - echo ("Hola Mundo");
    - echo "Hola Mundo";

- echo ($saldo);
- echo (“Tienes “ . $edad . “ años”);
- echo(“El valor es “ . $numeros[4]);
- echo “El saldo del cliente “ . codCliente . “ es “ . $saldo;

utf8_encode().

- Codifica una cadena a formato UTF-8.
- Sintaxis:
  - utf8_encode (“cadena texto”);
- Ejemplos:
  - echo utf8_encode (“Vergüenza y camión”);

utf8_decode().

- Decodifica una cadena en formato UTF-8.
- Sintaxis:
  - utf8_decode (“cadena texto”);
- Ejemplos:
  - echo utf8_decode ("VergÃ¼enza y camiÃ³n.");

nl2br().

- Permite insertar saltos de línea con el carácter o secuencia de escape “\n” al usarla con funciones que permiten imprimir datos por pantalla.
- Sintaxis:
  - nl2br (“cadena de caracteres \n otra cadena de caracteres” o $variables);
- Ejemplos:
  - echo nl2br(“texto 1 \n”); //Muestra texto 1 y luego se da un salto de línea.
  - echo nl2br(“texto 1 \n texto2”); //Muestra texto 1 y luego texto 2 en la siguiente línea.
  - echo nl2br(“texto 1 \n $numero”); //Muestra texto 1 y luego el contenido de la variable $numero en la siguiente línea.

print().

- Función que permite imprimir una cadena de caracteres y/o el contenido de variables, constantes, Arrays, etc.
- No son necesarios los paréntesis.
- Sintaxis:
  - print (“Texto”);
  - print ($variable);
  - print (“texto $variable”);
  - print (“texto“ . $variable);
  - print (‘texto’ . $variable);
- Ejemplos:
  - print (“Hola Mundo”);
  - print ($saldo);

- print (“Tienes “ . $edad . “ años”);
- print (“El valor es “ . números[4]);
- print (“El saldo del cliente “ . codCliente . “ es “ . $saldo);

print_r().

- Muestra el contenido de variables o arrays.
- Sintaxis:
  - print_r($variable);
  - print_r($array);
- Ejemplos:
  - $numeros = array (10,20,30);
  - $texto = “Marte”;
  - print_r($numeros); // Muestra [0]=>10, [1]=>20, [2]=>30.
  - print_r($texto); // Muestra Marte.

printf().

- Función que permite imprimir una cadena de caracteres y/o el contenido de variables, constantes, Arrays, etc.
- Permite dar formato a la información de salida.
- No son necesarios los paréntesis.
- Sintaxis:
  - printf (“formato”, dato);
- Formatos:
  - **%d.**
    - Muestra una cifra como decimal, positiva, negativa o cero.
  - **%c.**
    - Muestra un carácter al especificarse el valor decimal de su código ASCII.
  - **%o.**
    - Muestra el valor octal de una cifra.
  - **%x.**
    - Muestra el valor hexadecimal de una cifra en minúsculas.
  - **%X.**
    - Muestra el valor hexadecimal de una cifra en mayúsculas.
  - **%b.**
    - Muestra un número entero en formato binario.
  - **%e.**
    - Muestra número con notación científica en minúsculas.
    - Ejemplo de notación científica:
      - 1000000 => $1x10^6$ => 1E+06
      - 0,000001 => $1x10^{-6}$ =>1E-06
      - 56,89E+123 (Ejemplo de número muy grande).
  - **%E.**
    - Muestra número con notación científica en mayúsculas.
  - **%[.numero]s.**

        - Muestra el número de caracteres de una cadena que se especifique.
    - **%[.numero]f.**
        - Muestra el número de decimales que se especifique.
        - Si es necesario la cifra se redondea.
        - Si no se especifica número, muestra 6 posiciones decimales.
- Ejemplos:
    - $numero = 30.6568;
    - $texto = “Toledo y Segovia”;
    - printf (%.4s, $texto); // Muestra Tole.
    - printf (%.2f, $numero); // Muestra 30.66.
    - printf (%e, $numero); // Muestra 3.065680e+1.

var_dump().

- Muestra el contenido y el tipo de una variable o array.
- Sintaxis:
    - var_dump($variable);
    - var_dump($array);
- Resultado:
    - tipo (dato).
    - tipo (cantidad de caracteres) “dato”. (Con cadenas de caracteres).
- Ejemplos:
    - $numero = 30.6;
    - $texto = “Toledo”;
    - var_dump($numero); // Muestra float (30.6)
    - var_dump($texto); // Muestra string (6) “Toledo”

OPERADORES.

Aritméticos.

- Suma y signo positivo: +
- Resta y Signo Negativo: -
- Multiplicación: *
- División: /
- Potencia: **
    - Base**Exponente
    - Ejemplo:
        - 8 al cubo se escribiría como 8**3. Equivalente a 8 * 8 *8.
- Módulo o resto: %
- Opuesto/Negativo: -
- Ejemplos:
    - $a = 20;
    - $b = 10;
    - $c = 2;
    - echo $a + $b => 30
    - echo $a - $b => 10
    - echo $a * $b => 300
    - echo $a / $b => 2
    - echo $b ** $c => 100

- echo $a % $b => 0
- echo -$c => -2

Operadores para strings.

- **Concatenación.**
  - Operación que permite unir cadenas de caracteres ente si, o cadenas con variables, constantes, Arrays, etiquetas HTML, etc.
  - Se usa el operador punto (.)
  - Sintaxis:
    - "cadena1" . "cadena2";
    - "cadena" . $variable;
    - "<etiqueta HTML>Cadena</etiqueta HTML>" . $variable o $array o CONSTANTE;
  - Ejemplos:
    - echo "Te llamas " . $nombre;
    - echo "Te llamas " . $nombre . " " . $apellidos . " y tienes " . $edad . " años" ;
    - echo " Párrafo HTML concatenado a variable " . $resultado;
    - echo ("Resultado: " . $datos[4]);

Asignación.

- Sirven para asignar un valor a una variable, constante u otro tipo de estructuras de datos.
- Asigna el valor situado a la derecha del igual, al operando situado a la izquierda.
- Se usa el operador igual (=).
- **Tipos**:
  - **Asignación por valor**.
    - Asignación clásica, es decir, asignar un valor a una variable, etc.
    - Sintaxis:
      - nombre operando = valor;
    - Ejemplos:
      - $precio = 200.50;
      - $edad=20;
      - $nombre = "Cristina";
  - **Asignación por referencia**.
    - Permite que una variable haga referencia a otra variable.
    - Las 2 apuntan al mismo espacio de memoria y la modificación de una cambia la otra.
    - Se usa el operador &.
    - Sintaxis:
      - $variable1 = &$variable2;
    - Ejemplos:
      - $a = "hola";
      - $b = &$a;
      - echo $b; //Muestra "hola".
      - $a = "adiós";
      - echo $b; //Muestra "adiós".

Asignación sobre concatenación.

- Añade a la cadena del lado derecho, la cadena del lado izquierdo.
- Operador .=

- Sintaxis:
  - $a .= expresión; equivale $a = $a . expresión;
- Ejemplo:
  - $a = “hola;
  - $b = ” amigo”;
  - echo $a .= $b; // Devuelve “hola amigo”.
  - echo $a = $a . $b; // Devuelve “hola amigo”. Equivale a la anterior.

Operadores combinados.

- Combinación de operadores aritméticos y de asignación, o de concatenación y asignación.
- Formados por un operador aritmético seguido inmediatamente del operador de asignación.
- El primer operando debe ser una variable que almacenara el resultado y como segundo cualquier dato (variables, número, otra operación, etc.).
- Sintaxis:
  - +=, -=, *=, /=, **=, %=,
- Ejemplos:
  - $a += 5; equivale $a = $a+5;
  - $a += $b; equivale $a = $a+$b;
  - $a += (5*40); equivale $a = $a+(5*40);
  - $a -= 5; equivale $a = $a-5;
  - 4a *= 5; equivale $a = $a*5;
  - $a /= 5; equivale $a = $a/5;
  - $a %= 5; equivale $a = $a%5;
  - $a **= 5; equivale $a = $a**5;

Incremento/Decremento.

- **Incremento (++).**
  - Suma una unidad al operando.
  - Sintaxis:
    - $variable ++; equivale a $variable = $variable + 1 o también a $variable += 1;
  - Ejemplo:
    - $numero = 1;
    - $numero++; equivale a $numero = $numero + 1; $numero vale 2.
    - $numero++; equivale a $numero = $numero + 1; $numero vale 3.
  - El operador se puede poner delante o detrás de la variable.
    - **Preincremento**:
      - Primero se incrementa la variable y luego se lee el valor o se hace una operación.
      - Sintaxis:
        - ++$variable;
    - **Postincremento**:
      - Primero se lee el valor o se hace una operación, y luego se incrementa la variable.
      - Sintaxis:
        - $variable++;
      - Ejemplos:
        - $a = 5;
        - $a++; Devuelve 5 y luego pasa su valor a ser 6.
        - ++$a; Devuelve 6. Primero incrementa y se muestra el valor.

- **Decremento (- -).**
  - Resta una unidad al operando.
  - Sintaxis:
    - $variable--; $equivale a $variable =$variable -1;
  - Ejemplos:
    - $numero = 10;
    - $numero--; $equivale a $numero = $numero - 1; $numero vale 9.
    - $numero--; 4equivale a $numero = $numero - 1; $numero vale 8.
  - El operador se puede poner delante o detrás de la variable.
    - **Predrecremento**:
      - Primero se decrementa la variable y luego se lee el valor o se hace una operación.
      - Sintaxis:
        - --$variable;
    - **Postdecremento**:
      - Primero se lee el valor o se hace una operación, y luego se decrementa la variable.
      - Sintaxis:
        - $variable--;
      - Ejemplos:
        - $a = 5;
        - $a--; Devuelve 5 y luego pasa su valor a ser 4.
        - --$a; Devuelve 4. Primero decrementa y se muestra el valor.

Relacionales o de comparación.

- Permiten comparar los valores de 2 operandos y devuelven un valor booleano (***true*** - ***false***).
- Se utilizan para crear condiciones simples.
- Operadores:
  - > mayor que.
  - < menor que.
  - >= mayor o igual.
  - <= menor o igual,
  - == igual.
  - != distinto, diferente o lo contario.
  - === igualdad y tipos idénticos.
  - !== diferente y tipos diferentes.
- Ejemplos:

  var a = 23; a > 7; Verdadero o true.
  var b = “f”; var h = 3; b == h; Falso o false.
  var c = “Adiós”; c != “Hola”; Verdadero o true.
- Igualdad y tipos idénticos (===).
  - Se compara si los datos son iguales o ambos tipos coinciden.
  - Si tipo y dato coinciden devuelven true, sino false.
  - Ejemplo:
    - $a = “20”;
    - $b = 20;
    - $c = 5;
    - $d = 20;
    - $b===$c // false (Valores diferentes, tipos iguales).
    - $a===$b // false. (Valores iguales, tipos diferentes).

        - $b===$d // true. (Valores iguales, tipos iguales).
- Diferente y tipos diferentes (!==).
    - Se compara si los datos no son iguales y ambos tipos no coinciden.
    - Si tipo y dato no coinciden devuelven true, sino false.
    - Ejemplo:
        - $a = “20”;
        - $b = 20;
        - $c = 5;
        - $d = 20;
        - $b!==$c // false (Valores diferentes, tipos iguales).
        - $a!==$b // false. (Valores iguales, tipos diferentes).
        - $a!==$c // true. (Valores diferentes, tipos diferentes).

Lógicos.

- Sirve para crear condiciones compuestas.
- Devuelven verdadero o falso (true-false).
- Operadores:
    - and o && (Y lógico).
        - Para que se cumpla la condición global y se obtenga un valor verdadero, deben cumplirse obligatoriamente todas las parciales, si no, se produce un resultado falso.
        - Ejemplo:
            - ($a==5) && ($c>$r) && ($h!=”PHP”) // Devuelve ***true*** si todas las condiciones se cumplen obligatoriamente.
    - or o || (O lógico).
        - Para que se cumpla la condición global y se obtenga un valor verdadero, deben cumplirse al menos una condición parcial, si no, se produce un resultado falso.
        - Ejemplo:
            - ($a==5) || ($c>$r) || ($h!=”PHP”) // Devuelve ***true*** si al menos una condición se cumplen.
    - ! (No lógico o negación).
        - Invierte el valor del operando que le precede.
        - Devuelve verdadero si la expresión no es cierta.
        - Ejemplo:
            - $x = ***false***;
            - $y = !x;
            - echo $y; // Devuelve ***true***.
    - xor (O lógico exclusivo).
        - Devuelve ***true*** si una condición parcial es verdadera y la otra falsa.
        - Devuelve ***true*** si una condición parcial es verdadera, pero no ambas.
        - Ejemplo:
            - $x = 23;
            - $y = 47;
            - ($x==23) xor ($y>50) // Devuelve ***true*** porque una condición se cumple, pero no la otra.
- Las diferencias entre and y &&, y entre or y || es sólo de prioridad, siendo menor en and y en or.

Comparación combinada u operador nave espacial.

- Permite comparar 2 variables, constantes, literales, Arrays objeto o expresiones en general.

- Operador: < = >
- Devuelve, tras realizar la comparación:
  - **-1:**
    - La primera variable o expresión es menor que la segunda.
  - **0:**
    - La primera variable o expresión es igual que la segunda.
  - **1:**
    - La primera variable o expresión es mayor que la segunda.
- Sintaxis:
  - Expresión 1 < = > Expresión 2;
  - $variable 1 < = > $variable 2;
- Ejemplos:
  - $a = 20;
  - $b = 20;
  - $c = 5;
  - $texto = “Hola”;
  - echo $a < = > $b; // Muestra 0.
  - echo $a < = > $c; // Muestra 1.
  - echo $c < = > $a; // Muestra -1.
  - echo $texto < = > ”Hola”;// Muestra 0.
  - echo [1, 2, 3] <=> [1, 2, 4]; // Muestra -1.

Operador de coalescencia nulo.

- También denominado unión null.
- Permite asignar a una variable un valor u otro en función de si éstos existen.
- Operador: ??
- Sintaxis:
  - $variable = expresión1 ?? Expresión 2;
- Ejemplos:
  - $a =20;
  - $b = $c ?? “Hola”; // Se guarda en la variable la palabra “Hola” porque $c no existe.
  - $b = $a ?? $c; // Se guarda en la variable el valor de la variable $a, que si existe y tiene un valor cargado.

Operador ternario.

- Permite crear una condición completa.
- Sintaxis:
  - operando 1? operando 2: operando 3;
    - Operando 1 es una condición.
    - Operando 2 es el valor si verdadero.
    - Operando 3 es el valor si falso.
- Ejemplo:
  - $edad =20;
  - $mayoriaEdad = $edad >=18? “mayor de edad”: “menor de edad”;
  - echo “Eres “ . $mayoriaEdad;
  - echo $edad >=18? “mayor de edad”: “menor de edad”;
  - Ejemplo equivalente a:

```
if ($edad >=18)
{
        echo “mayor de edad”:
}
```

```
else
{
        echo “menor de edad”:
}
```

Operador de tipo instanceof.

- Se usa para saber si una variable es un objeto instanciado de una clase**.**
  - Sintaxis:
    - $variable instanceof Clase;
  - Ejemplo:

```
class Clase1
{
        Código de la Clase1;
}
class Clase2
{
        Código de la Clase2;
}
$a = new Clase1();
echo($a instanceof Clase1); // Devuelve ***true***.
echo($a instanceof Clase2); // Devuelve ***false.***
```

Operador de resolución de alcance (Paamayim Nekudotayim).

- Operador ::
  - Permite acceder o hacer referencia a:
    - Variables, constantes y métodos en el interior de las clases.
    - Métodos o constantes se encuentran dentro de clases, pero de las cuales no se han creado objetos para, a través de ellos y del operador flecha (->), acceder a dicho métodos.
- Sintaxis:
  - Dentro de clases:
    - self::variable de clase;
    - self::constante de clase;
    - self::método de clase o estático();
  - Fuera de clases sin crear objetos:
    - $nombre de clase::constante de clase;
    - $nombre de clase::método de clase o estático();
  -
- Ejemplo:

```
class miClase
{
        public $A = "Algo";
        public static $B = 40;
        public const C = "PHP";
        public static metodo1()
        {
                echo $this::B . "<br>";
                        echo "Te llamas: " . $this->nombre . "<br>";
```

```
                    echo $this::B . "<br>";
        }
        public metodo2()
            {
                    echo "Te llamas: " . $this->nombre . "<br>";
            }
        }
        $a = new miClase();
        $a->metodo2();// Al no ser estático necesario crear un objeto
        para acceder a metodo2() usando el operador acceso a
        miembros de una clase ->.
        miClase:: metodo1(); // Método estático o de clase. No necesario
        usar objetos para llamarlo.
        $a->metodo1();// También es válido.
        echo $a->A  . "<br>"; // Válido al ser $A una variable dinámica.
        echo $a::B . "<br>"; //Error al ser $B una variable estática. Uso
        del operador de resolución de alcance.
        echo miClase::B . "<br>"; // Válido al ser $B una variable de clase
        o estática. Uso del operador de resolución de alcance.
```

Operador flecha (->).

- Se usa para acceder a los miembros de una clase (variables y métodos), a través de un objeto o instancia de ésta.
- Sintaxis:
  - objeto -> propiedad;
  - objeto -> método ();
- Ejemplos:
  - cliente1 -> $cod_cuenta;
  - cliente1 -> $nombre;
  - cliente1 -> modificarDatos();

Precedencia o prioridad de operadores.

- En una operación en la que estén implicados varios operadores distintos, unos actuaran antes que otros.
- La prioridad se puede modificar usando paréntesis, de modo que, lo que se encierre entre paréntesis tiene la máxima prioridad.
- Ejemplos**:**
  - 7+5**2= 32
  - (7+5)**2=144
  - 7+9+5/3=17,666 // Media incorrecta.
  - (7+9+5)/3=7
- **Prioridades**: (de menor a mayor prioridad).

```
or
xor
and
=  +=  -=  *=  /=  %=  **=  .=
?:
??
||
&&
```

```
== != === < = >
< <= > >=
+ - .
* / ** %
! ++ -- (int) (double) (string) (array) (object)
```

INSTRUCCIONES ALTERNATIVAS, DE BIFURCACIÓN O CONDICIONALES.
Condiciones:

- **Condición simple.**
  - Se utilizan dos operandos y un operador de comparación.
  - Se pueden comparar diversos operandos:
    - $a > 8 // Variable con valor.
    - $a == $b // Variable con otra variable.
    - $a != ”Vacaciones” // Variable con un texto.
- **Condición compuesta:**
  - Se evalúan varias condiciones simples mediante operadores lógicos.
  - Y lógica (&& o and)
    - Para que se cumpla la condición global y se realicen las acciones deben cumplirse obligatoriamente todas las condiciones simples.
    - Ejemplo:
      - (($a>$b) && ($b>$c))
      - ($a>8 && $c==$d && $e<6)
      - (($a>$b) and ($b>4c))
  - O lógica (|| u or)
    - Para que se cumpla la condición global y se realicen las acciones deben cumplirse al menos una condición simple.
    - Ejemplo:
      - (($a>b) || ($b>$c))
      - ($a>8 || $c==$d || $e<6)
      - (($a>b) or ($b>$c))
  - Combinación de operadores lógicos.
    - Ejemplo:
      - ($a>8 && ($c==$d || $e<6))

Estructuras condicionales:

Condicional simple. (if)

- Sólo se ejecutan las instrucciones si la condición es verdadera, sino no se hace nada.
- Sintaxis:

```
if (condición simple o condición compuesta)
{
        instrucciones;
}
```

```
otras instrucciones;
```

- Ejemplo:

```
$a = 7;
$b = 5;
if ($a>$b)
{
        echo “$a es mayor que $b”;
}
```

Condicional completa. (if…else)

- Se ejecutan unas instrucciones si la condición es verdadera y si no, otras instrucciones.
- Sintaxis:

```
If (condición simple o condición compuesta)
{
        instrucciones;
}
else
{
        otras instrucciones;
}
```

- Ejemplo:

```
$a = 7;
$var $b = 5;
if ($a>$b)
{
        echo (“$a es mayor que $b”);
}
else
{
        echo (“$a no es mayor que $b”);

}
```

Condicional anidada. (if…else…if)

- Se evalúan varias condiciones en cadena.
- Si se cumple una condición se realizan unas acciones, si no, se pasa a evaluar la siguiente condición.
- Si no se cumple ninguna condición se pueden o no realizar también acciones.
- Sintaxis:

```
if(condición 1)
{
        instrucciones;
}
else if(condición 2)
```

```
{
        instrucciones;
}
else if(condición N)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple ninguna condición.; // (opcional)
}
```

- Ejemplo:

```
$a = 5;
$b = 8;
if($a>$b)
{
        echo "$a es mayor que $b";
}
else if($a==$b)
{
        echo "$a es igual que $b";
}
else if($a<$b)
{
        echo ("$a es menor que $b");
}
```

- Otro ejemplo con else final:

```
$a = 5.6;
$b = 8.9;
if($a>$b)
{
        echo "$a es mayor que $b";
}
else if($a==$b)
{
        echo "$a es igual que $b";
}
else
{
        echo ("$a es menor que $b");
}
```

Condicional múltiple sin anidamiento. (if… …if)

- Se evalúan varias condiciones independientes.
- Si no se cumple ninguna condición se pueden o no realizar también acciones.
- Sintaxis:

```
if(condición 1)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple la condición.; // (opcional)
}

if(condición 2)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple la condición.; // (opcional)
}

if(condición N)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple la condición.; // (opcional)
}
```

- Ejemplo:

```
$a = 150;
if($a>=100 && $a <=200)
{
        echo “$a está entre 100 y 200”;
}
else
{
        echo “$a no está en el intervalo”;
}

if($a==150)
{
        echo “$a es igual a 150”;
}
if($a>=201 && $a <=2000)
{
```

```
        echo "$a está entre 201 y 2000";
    }
```

Condicional múltiple. (switch…case)

- Permite evaluar varias condiciones a la vez, que pueden ser valores fijos en una variable, rangos o condiciones.
- Se pueden agrupar varios case cuando con todos ellos se vayan a realizar la misma acción, que se incluye al final del último que forme el grupo.
- Sintaxis:

```
switch (expresión)
{

    case valor 1:
        instrucciones;
        break;

    case valor 2:
        instrucciones;
        break;

    case valor N:
        instrucciones;
        break;

    default:
        instrucciones;
        break;
}
```

- Expresión:
  - Puede ser una variable, una expresión matemática, valor booleano, …
- Valor:
  - Puede ser un texto, un número o una condición simple o compuesta entre paréntesis.
  - También se pueden usar operadores relacionales y lógicos:
    - case ($a > $b):
    - case ($a >$ b && $b != $c):
  - Para utilizar operadores relacionales y lógicos como expresión, en switch se debe incluir un valor booleano (true o false) o una variable a evaluar.
    - switch(true) {}
    - switch($precio) {}
- break:
  - Opcional.

- Si se cumple un valor se sale de la instrucción switch.
- Si no se pone, se continúan evaluando otros valores.

- default:
  - Opcional.
  - Instrucciones que se realizarán si el valor evaluado no coincide con ningún case.
- Ejemplos:
  - Con variable y valores textuales.

```
$color;
switch ($color)
{

        case "verde":
                echo "El $color es muy bonito";
                break;

        case "negro":
                echo "El $color no me gusta";
                break;

        default:
                echo "Prefiero otro color";
                break;
}
```

  - Con variable y valores textuales agrupados.

```
$color;
switch ($color)
{

        case "verde":
        case "azul":
        case "rojo":

                echo "El $color es muy bonito";
                break;

        case "negro":
        case "marrón":

                echo "El $color no me gusta";
                break;

        default:
                echo "Prefiero otro color";
```

```
                    break;
}
```

- Con valor booleano y condiciones.

```
$n1 = 67;
$n2 = 56;
switch(true)
{
        case ($n1 > $n2):
                echo ("El primer número es mayor que el segundo");
                break;
        case ($n1 == $n2):
                echo ("Ambos números son iguales");
                break;
        case ($n1<$n2):
                echo ("El segundo número es mayor que el primero");
                break;
 }
```

- Con variable a evaluar y condiciones.

```
$mes = “Julio”;
switch($mes)
{
        case ($mes == “Enero”):
                echo "Invierno";
                break;
        case ($mes == “Julio”):
                echo "Verano";
                break;
        case $mes == “Octubre”:
                echo "Otoño";
                break;
 }
```

INSTRUCCIONES REPETITIVAS O BUCLES.
Mientras - While.

- Se ejecutan un conjunto de instrucciones en bucle mientras se cumpla una condición.
- Cada vuelta en el bucle se denomina iteración.
- Si no se cumple inicialmente la condición no se entra en el bucle.
- Puede funcionar con una variable contadora o no.
- Sintaxis:

```
while (condición)
{
        instrucciones;
```

```
}
```

- Ejemplos**:**
  - **Con variable contadora.**

```
$a = 0;
while($a<=9)
{
        echo ($a);
        $a++  /$a=$a+1 / $a+=1;  // Incrementos de unidad en
        unidad.
        $a=$a+5 / $a+=5; // incrementos de 5 en 5.
}
echo “Fin del bucle $a”;
```

  - **Con condición a cumplir.**

```
while($b != “hola”)
{
        echo ($b);
        if($hoy == “lunes”)
        {
        $b = “hola”;
        }
}
echo “Fin del bucle $b”;
```

Hacer-Mientras o Repetir-Hasta o Do-While.

- Se ejecutan un conjunto de instrucciones en bucle mientras se cumpla una condición.
- Cada vuelta en el bucle se denomina iteración.
- Si no se cumple inicialmente la condición al menos se realizan una vez las instrucciones.
- Puede funcionar con una variable contadora o no.
- Sintaxis**:**

```
do
{
        instrucciones;
} while (condición);
```

- Ejemplos**:**
  - **Con variable contadora.**

```
$a = 0;
do
{
        echo ($a);
```

```
        $a++ ; /$a=a+1 / $a+=1  // Incrementos de unidad en
unidad.
        //$a=$a+5 / $a+=5 // incrementos de 5 en 5.
} while($a<=9);
echo “Fin del bucle $a”;
```

- **Con condición a cumplir.**

```
$b = “adios”;
do
{
        echo ($b);
        if($hoy == “lunes”)
        {
        $b = “hola”;
        }
} while($b != “hola”);
echo “Fin del bucle $b”;
```

Bucle for.

- Se repiten unas instrucciones un número determinado de veces hasta que deja de cumplirse una condición.
- La condición está determinada por una variable contadora.
- Se pueden usar varias variables contadoras para anidar un bucle for a otro.
- Sintaxis:

```
for(variable contadora = valor inicial; condición; incremento o decremento)
{
        Instrucciones;
}
```

  - **Valor inicial**:
    - Valor inicial de la variable contadora que puede crearse antes del bucle o en el momento de asignarle el valor inicial.
  - **Condición**:
    - Coincide con el valor final de la variable contadora.
    - Evalúa en cada vuelta si el valor de la variable contadora cumple la condición, si no, se finaliza el bucle.
    - Si la condición es falsa desde el principio no se ejecuta el bucle.
  - **Incremento/decremento**:
    - Permite actualizar la variable contadora con el fin de hacer que llegue a ser falsa la condición en algún momento.
- Ejemplos:
  - Recorrido por una variable:

```
        for($x = 1;  $x <= 10; $x++)
        {
```

```
            echo 'La variable $x vale:  ' . $x);
        }
```

- Recorrido de un array:

```
        $colores = array("rojo","verde","azul");
        $cantidad = 3;
        //$cantidad = sizeof ($colores);
        for($x = 0; $x < $cantidad; $x++)
        {
          echo "Posición [$x], valor:  $colores[$x]";
        }
```

- Dos bucles anidados.

```
        for($x = 1;  $x <= 10; $x++)
        {
            for($y = 1;  $y <= 10; $y++)
            {
                echo 'Valor de $x: ' . $x . 'valor de $y: ' . $y;
            }
        }
```

- **Sentencias break y continue**.
  - Se usan en bucles y estructuras condicionales para finalizar la repetición de instrucciones o terminar las condiciones.
  - **break**.
    - Sirve para finalizar prematuramente un bucle.
    - Se incluye en una condición dentro del bucle que hace que este termine.
    - Ejemplo:

```
        $x;
        for($x = 1;  $x <= 10; $x++)
        {
            if ($x==5)
            {
                break;
            }
           echo "la variable contadora vale: " . $x;
        }
```

  - **continue**.
    - Permite saltar una iteración de un bucle, ignorando las instrucciones que hubiera a continuación.
    - Tras el salto el bucle continúa funcionando hasta que termine de forma normal.
    - Ejemplo:

```
        $x;
        for($x = 1;  $x <= 10; $x++)
        {
            if ($x==5)
```

```
        {
            continue;
        }
    echo "la variable contadora vale: " . $x;
}
```

SINTAXIS ALTERNATIVA PARA INSTRUCCIONES DE CONTROL.

- Para simplificar instrucciones condicionales y/o repetitivas, en PHP se pueden usar instrucciones abreviadas.
- En ellas se sustituye la primera llave ({), por 2 puntos (:), y la llave de cierre (}) por las cláusulas endif, endwhile, endfor, endforeach y endswitch, según la instrucción utilizada.
- Sintaxis:

  - **If simple.**

```
if (condición):
Instrucciones si verdadero;
endif;
```

  - **If completo.**

```
if (condición):
Instrucciones si verdadero;
else:
Instrucciones si falso;
endif;
```

  - **If anidado.**

```
if (condición 1):
Instrucciones;
elseif (condición 2):
Instrucciones;
elseif (condición N):
Instrucciones;
else: // (Este else es opcional).
Instrucciones si no se cumple lo anterior;
endif;
```

  - **switch…case.**

```
switch(expresión):
```

```
case valor 1:
Instrucciones;
break;
case valor 2:
Instrucciones;
break;
case valor N:
break;
Instrucciones;
default: // (Este default es opcional).
Instrucciones si no se cumple lo anterior;
break;
endswitch;
```

- **for.**

```
for (valor inicial variable contadora; condición;
incremento/decremento):
Instrucciones;
endfor;
```

- **foreach.**

```
foreach ($array u $objeto as [$clave =>] $valor):
Instrucciones;
endforeach;
```

- **while.**

```
while(condición):
Instrucciones;
endwhile;
```

ARRAYS.

- Estructura que permite almacenar un conjunto de datos tales como números, cadenas de caracteres, valores booleanos, otros arrays, imágenes, objetos, etc.
- Otros Nombres:
  - Vector, matriz, tabla o matriz unidimensional, lista o arreglo.
- Composición:
  - **Elementos**.
    - Cada dato del array.
  - **Índices**.
    - Hace referencia a la posición en que hay un dato en el array o un hueco donde almacenar un dato.
    - Primer índice = 0.
    - Último índice = N-1.
- **Creación de un Array.**

- **Arrays indexados:**
  - **Función array().**
    - Crea un array con tantas posiciones como valores se hayan incluido como parámetros.
    - Cada posición o hueco se rellenará con el valor especificado en el mismo orden.
    - Sintaxis:
      - $nombre del array = array(valor1, valor2,…,valor N);
    - Ejemplo:
      - $colores = array(“Rosa”, ”Naranja”, ”Granate”);
  - **Asignando valores a cada índice.**
    - E l array se crea al asignar el primer valor.
    - Sintaxis:
      - $nombre del array[número] = valor;
    - Ejemplo:
      - $precios[8] = 200.00; // Se crea el array con 8 posiciones, las 7 primeras vacías.
      - $precios[1] = 100.00;// Dato en posición vacía ya creada anteriormente.
  - **Asignando valores sin usar índice.**
    - E l array se crea al utilizar la primera vez la instrucción.
    - Los valores se van colocando automáticamente en orden correlativo empezando por el índice cero (0).
    - Sintaxis:
      - $nombre del array[] = valor;
    - Ejemplo:
      - $precios[] = 200.00; // Creación del array y valor 200.00 a la posición 0.
      - $precios[] = 100.00;// Valor 100.00 a la posición 1.
  - **Asignando valores entre corchetes.**
    - Se usan corchetes que encierran los datos en el mismo orden que ocuparán en las posiciones del array.
    - Sintaxis:
      - $nombre del array = [valor1, valor2,…,valor N];
    - Ejemplo:
      - $colores = [“Rojo”, ”Verde”, ”Azul”];
- **Arrays asociativos:**
  - Creación de índices propios numéricos y/o alfanuméricos.
  - Se pueden crear índices personalizados a la par que en ellos se introduce un valor.
  - En un array asociativo cada elemento del array se compone de un valor y una clave:
    - La clave define la posición o índice.

        - El valor representa al dato o elemento introducido o asociado a una clave.
    - Para crearlos se puede usar la función array(), asignando valores mediante corchetes o asignando valores cada índice.
    - Índice entre comillas si es una cadena de texto.
    - Se accede a los elementos contenidos en el array por su índice o clave asociativa.
        - Sintaxis:
            - $nombre del array = array(clave o índice 1=> valor1, clave o índice 2=> valor2,…,clave o índice N=> valorN);
            - $nombre del array = [clave o índice 1=> valor1, clave o índice 2=> valor2,…,clave o índice N=> valorN];
            - $nombre del array[“clave o índice1”] = valor1;
              $nombre del array[“clave o índice2”] = valor2;
              $nombre del array[“clave o índiceN”] = valorN;
        - Ejemplos:
            - $numeros1 = array(“cero”=> 20, “uno”=> 40,”dos”=>60);
            - $colores_coches = array(“Ibiza” => “Rojo”, ,”Fiesta” =>”Plateado”, ”Sandero”=> “amarillo”);
            - $numeros2 = [“cero”=> 20, “uno”=> 40,”dos”=>60];
            - $numeros3 [“cero”]= 20;
              $numeros3[“uno”]= 40;
              $numeros3[”dos”] = 60;

- **Arrays multidimensionales.**

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]
[2][0]	[2][1]	[2][2]	2][3]	[2][4]	[2][5]

    - Array de Arrays.
    - Array en el cual cada uno de sus elementos puede ser un array, y así sucesivamente.
    - Creación:
        - **(Sintaxis y ejemplos para Arrays de 2 dimensiones, pero siguen la misma lógica que para Arrays de 3, 4 o más dimensiones).**
        - **Creación como array indexado y función array().**
            - Sintaxis.
                - $nombre del array = array
                  (
                  array(valor1, valor2,…, valorN),
                  array(valor1, valor2,…, valorN),
                  array(valor1, valor2,…, valorN)
                  );
            - Ejemplo:

```
$productos =  array
(
    array(100, “TV”, 400.60),
    array(200, “DVD”, 125.00),
    array(300, “TV”, 650.00)
);
```

- **Creación como array asociativo y función array().**
  - Sintaxis.

```
$nombre del array =  array
(
    array(clave o índice 1 => valor1, clave o índice 2 =>
    valor2,…, clave o índice N => valorN),
    array(clave o índice 1 => valor1, clave o índice 2 =>
    valor2,…, clave o índice N => valorN),
     array(clave o índice 1 => valor1, clave o índice 2 =>
     valor2,…, clave o índice N => valorN)
);
```

  - Ejemplo:

```
$productos =  array
(
    array(“código” => 100, “producto => “TV”,
    “precio” => 400.60),
    array((“código” => 200, “producto => “DVD”,
    “precio” => 125.00),
    array((“código” => 300, “producto => “TV”,
    “precio” => 650.00)
);
```

- **Creación como array indexado con índices, corchetes y función array.**
  - Sintaxis.

```
$nombre del array [índice 1] =  array (valor1,
valor2,…,valor N);
$nombre del array [índice 2] =  array (valor1,
valor2,…,valor N);
$nombre del array [índice N] =  array (valor1,
valor2,…,valor N);
```

  - Ejemplo:

```
$productos [0] = array(100, “TV”, 400.60);

$productos [1] = array(200, “DVD”, 125.00);

$productos [2] = array(300, “TV”, 650.00);
```

- **Creación como array asociativo con índices, corchetes y función array.**
  - Sintaxis.
    - $nombre del array [índice 1] = array (clave o índice 1 => valor1, clave o índice 2 => valor2,…, clave o índice N => valor N);
      $nombre del array [índice 2] = array (clave o índice 1 => valor1, clave o índice 2 => valor2,…, clave o índice N => valor N);
      $nombre del array [índice N] = array (clave o índice 1 => valor1, clave o índice 2 => valor2,…, clave o índice N => valor N);

  - Ejemplo:
    - $productos [0] = array((“código” => 100, “producto => “TV”, “precio” => 400.60);

      $productos [1] = array(((“código” => 200, “producto => “DVD”, “precio” => 125.00);

      $productos [2] = array(((“código” => 300, “producto => “TV”, “precio” => 650.00);

Acceso a un Array.

- **Introducir elementos en un Array.**
  - Si en una posición hay un dato se sustituye por el nuevo.
  - Si no hay un elemento se carga en la posición especificada.
    - Sintaxis:
      - $nombre del array[índice] = valor;

    - Ejemplos:
      - $letras = array(“a”, “y”, “f”, “v”, “e”, “h”, “b”, “i”);
      - $letras[0] = ”z”; // “z” sustituye a “a”.
      - $letras[8] = “g”; // Nueva letra a continuación de la última.
      - $productos[“código”] = 120; // Nuevo código.
- **Eliminar elementos de un Array.**
  - Se asigna un valor nulo a su contenido.
  - No se borra la posición, sólo el contenido.
  - Sintaxis:
    - nombre del array[índice] = null;
    - nombre del array[clave asociativa] = null;
  - Ejemplo:
    - $colores = array (“Rojo”, ”Verde”, ”Azul”);
    - $colores[1] = null;
    - echo sizeof ($colores); // Mostrará que hay 3 posiciones, aunque la segunda esté vacía por haber eliminado el color verde.

    - $productos[“precio”] // Elimina el precio en un array asociativo.
- **Mostrar elementos individuales de un Array.**
    - Para mostrar los elementos de un array u operar con su contenido se puede acceder a posiciones determinadas del array especificando un índice o una clave en un array asociativo.
    - Sintaxis:
        - nombre del array[índice];
        - nombre del array[clave o índice asociativo];
    - Ejemplos:
        - **Array indexado**:
            - $colores = array (“Rojo”, ”Verde”, ”Azul”);
            - $numeros = array (1, 6, 8, 3, 4, 9, 5);
            - echo “El primer color de la lista es: $colores[0]);
            - $color1 = colores[0];
            - $suma = numeros[1] + numeros[4];
        - **Array asociativo**:
            - $capitales = array (“España” => “Madrid”, (“Francia” => “París”, (“Italia” => “Roma”);
            - echo “Capital del Francia: $capitales[“Francia]”;
        - **Array multidimensional indexado**.
            - Para mostrar el valor de una posición concreta hay que especificar los índices de los Arrays que convergen en ese dato.
            - Sintaxis:
                - $nombre del array[índice del array interno][índice del valor a mostrar];
            - Ejemplos:
                - echo $productos[0][1]; // Muestra TV.
                - echo $productos[1][2]; // Muestra 125.00, precio del DVD.
        - **Array multidimensional asociativo**.
            - Para mostrar el valor de una posición concreta hay que especificar los índices de los Arrays que convergen en ese dato, siendo el segundo índice, la clave asociativa.
            - Sintaxis:
                - $nombre del array[índice del array interno][índice asociativo o clave del valor a mostrar];
            - Ejemplos:
                - echo $productos[0][“producto”]; // Muestra TV.
                - echo $productos[1][“precio”]; // Muestra 125.00, precio del DVD.

Recorrer un array usando un bucle.

- **Bucle for.**

- Se usa la variable contadora, tanto para dar vueltas en el bucle, como para indicar los índices del array.
- Ideal para recorrer Arrays indexados, pero no para asociativos.
- Sintaxis:

```
for ($variable contadora o índice = 0; $variable contadora o índice < longitud del Array; $variable contadora o índice++)
{
    echo "nombre del array[índice]";
}
```

- Ejemplo:
- Con valor conocido para la condición del bucle:

```
$colores = array ("Rojo", "Verde", "Azul");
//$longitud = 3;
$longitud = sizeof($colores);
for ($i = 0; $i < $longitud; $i++)
// for ($i = 0; $i < sizeof($colores); $i++)

{
    echo "Contenido del array: $colores[$i]";
}
```

- **Bucle foreach.**
  - Permite recorrer Arrays y objetos sin preocuparse por su número de elementos.
  - Ideal para Arrays asociativos, aunque también sirve para indexados.
  - Sintaxis:
    - **Mostrar solo los valores:**

```
foreach($nombre del array as $variable valor)
{
    Instrucciones;
}
```

      - $variable valor:
        - Variable que recoge el valor del array en cada iteración.
    - Ejemplo:

```
$colores = array ("Rojo", "Verde", "Azul");
foreach($colores as $color)
{
    echo "Color incluido: " .$color;
}
```

    - **Mostrar valores y/o índices:**

```
foreach($nombre del array as $índice=>$variable valor)
{
    Instrucciones;
}
```

      - $índice:

- Variable que recoge el valor del índice del array en cada iteración.

- Ejemplo:

```
$colores = array (“Rojo”, ”Verde”, ”Azul”);
foreach($colores as $indice =>$color)
{
        echo “El color incluido en la posición $indice es el”
    .$color;
}
```

Recorrer un array multidimensional usando un bucle.

- **Bucle for.**
  - Se usan tantos bucles anidados como dimensiones tenga el array.
  - En el bucle más interior se incluyen las instrucciones que mostrarán los datos e índices de array, o que servirán para hacer otras operaciones.
  - Se usan las variables contadoras, tanto para dar vueltas en el bucle, como para indicar los índices del array en cada dimensión.
  - Ideal para recorrer Arrays indexados, pero no para asociativos.
  - Sintaxis:

```
for ($variable contadora o índice = 0; $variable contadora o índice < longitud del Array; $variable contadora o índice++) // 1ª dimensión.
{
    for ($variable contadora o índice = 0; $variable contadora o índice < longitud del Array; $variable contadora o índice++) // 2ª dimensión y así sucesivamente.
    {
        Instrucciones;
        echo “nombre del array[índice dimensión 1] [índice dimensión 2]”;
    }
}
```

  - Ejemplo:
  - Con valor conocido para la condición del bucle:
    -

```
$productos =  array
                (
                    array(100, “TV”, 400.60),
                    array(200, “DVD”, 125.00),
                    array(300, “TV”, 650.00)
                );
//$longitud= 3;

for ($i = 0; $i < $longitud; $i++)
{
    echo “Datos de la fila “. $i;
    for ($j = 0; $j < longitud; $j++)
```

```
    {
        echo "Posición " . $i. " " . $j . "Contenido del array: $colores[$i][$j]";
    }
}
```

- **Bucle foreach.**
  - Se usan tantos bucles anidados como dimensiones tenga el array.
  - En el bucle más interior se incluyen las instrucciones que mostrarán los datos e índices de array, o que servirán para hacer otras operaciones.
  - Permite recorrer Arrays y objetos sin preocuparse por su número de elementos.
  - Ideal para Arrays asociativos, aunque también sirve para indexados.
  - Sintaxis:
    - **Mostrar solo los valores:**
    - Sintaxis**:**
      -
```
foreach($nombre del array as $variable valor 1)
{
    foreach ($variable valor 1 as $variable valor 2)
    {
        Instrucciones;
        echo "Contenido: ". $variable valor 2;
    }
}
```
    - Ejemplo:
      -
```
$productos =  array
    (
        array("código" => 100, "producto => "TV",
        "precio" => 400.60),
        array(("código" => 200, "producto => "DVD",
        "precio" => 125.00),
        array(("código" => 300, "producto => "TV",
        "precio" => 650.00)
    );

    foreach($productos as $pro1)
    {
        foreach ($pro1 as $pro2)
        {
            echo "Contenido: ". $pro2;
        }
    }
```

    - **Mostrar valores y/o índices:**
    - Sintaxis:

```
foreach($nombre del array as $índice1=>$variable valor 1)
{
        foreach ($variable valor 1 as $índice2=> $variable valor 2)
        {
                Instrucciones;
                echo "Posición: " . $indice1 . " " .$indice2. "
                Contenido: ". $variable valor 2;
        }
}
```

- Ejemplo:

```
foreach($productos as $i=>$pro1)
{
        foreach ($pro1 as $j =>$pro2)
        {
                echo "Posición: " . $i . " " .$j. " Contenido:
                ". $pro2;
        }
}
```

Copiar un array.

- Para copiar un array sólo hay que crear un array nuevo asignándole el nombre del array a copiar.
- Sintaxis.
  - $nombre del array copia = $ nombre del array a copiar;
- Ejemplo:
  - $numeros = array (4,9,2,5,7);
  - $numeros_copia = $numeros;

Copiar un array usando el método getArrayCopy().

- Otra opción para copiar un array es usar el método getArrayCopy() de la clase ArrayObject.
- Sintaxis.
  - $nombre del objeto array = new ArrayObject($nombre array a copiar);
- Ejemplo:
  - $nuevoArraycolores = new ArrayObject($colores);
  - $colores_copia2 = $nuevoArraycolores->getArrayCopy**()**

FUNCIONES PARA ARRAYS.

- **count().**
  - Cuenta el número de elementos que contiene un array.
  - Sintaxis.

        - count ($nombre del array, modo);
        - modo:
            - Indica la forma en que la función contará los elementos contenidos en el array:
                - **COUNT_NORMAL** o **0**:
                    - Cuenta los elementos que hay en un array unidimensional o en el primer nivel de uno multidimensional.
                    - Valor por defecto.
                - **COUNT_RECURSIVE** o **1**:
                    - Cuenta los elementos que hay en un array multidimensional.
    - Ejemplos:
        - $numeros1 = array(4,9,2,5,7);
        - $numeros2 = array(array(4,9),array(5,7));
        - echo “Número de datos en el array: “ . count ($numeros1);
        - echo “Número de datos en el array: “ . count ($numeros2); // Muestra 2 elementos.
        - echo “Número de datos en el array: “ . count ($numeros2, 1); // Muestra 6 elementos.
- **sizeof().**
    - Cuenta el número de elementos que contiene un array.
    - Alias de count().
    - Sintaxis.
        - sizeof ($nombre del array);
    - Ejemplo:
        - $numeros = array(4,9,2,5,7);
        - echo “Número de datos en el array: “ . sizeof ($numeros);
- **implode().**
    - Muestra un listado de los valores que hay en un array intercalando el carácter separador que se le especifique.
    - Si no se incluye un separador, se muestran todos los datos juntos en la misma línea sin espacio que los separe.
    - Para comprobar su funcionamiento, hay que incluir esta función dentro de alguna de las funciones de salida de datos como echo (), print(), print_r(), var_dump(), etc.
    - Sintaxis:
        - implode(“separador”, nombre del array);
    - Ejemplo:
        - print (implode (“, “, $numeros)); // Muestra el contenido del array $numeros en una línea separando sus datos con una coma y un espacio en blanco.
        - print (implode (“
“, $numeros)); // Muestra cada dato del array $numeros en una línea distinta.
- **in_array().**
    - Busca dentro de un array el valor pasado como parámetro.
    - Devuelve un valor booleano. (***true***/***false***).

- Sintaxis.
  - in_array (valor buscado, $nombre del array);
- Ejemplo:

```
if( in_array(9, $numeros))
{
        echo “El dato está en el Array”;
}
else
 {
        echo “El dato no está en el Array”;
}
```

- **list().**
  - Asigna a las variables indicadas como parámetros en list(), los valores del array al que iguala.
  - No tiene porqué crearse el mismo número de variables que de elementos en el array.
  - Los valores se asignan en orden de creación de las variables siguiendo el índice 0.
  - No funciona con arrays asociativos directamente, hay que usar array_keys() o array_values(), para cargar claves asociativas o valores respectivamente.
  - Sintaxis.
    - list ($variable 1, $variable 2, …, $variable N) = $nombre array;
  - Ejemplo:
    - $numeros = array(4,9,2);
    - list ($num1, $num2, $num3) = $numeros;
    - list($n1, ,$n3) =$numeros;// El valor 9 no se carga en ninguna variable.
    - echo “$num1; // 1ª variable definida muestra 4, equivale a:
    - echo $numeros[0];
- **array_search().**
  - Permite buscar un elemento en un array y mostrar cuál es su índice (array indexado) o clave asociativa (array asociativo).
  - Sintaxis.
    - array_search (valor buscado, $nombre del array);
  - Ejemplo:
    - $numeros = array(4,9,2,5,7);
    - $producto = array(“código” => 100, “producto” => “TV”, “precio” => 400.60);
    - echo “Índice del elemento buscado: “. array_search(9, $numeros); // Mostrará el índice 1.
    - echo “Clave del elemento buscado: “. array_search(“TV”, $producto); // Mostrará la clave producto.
    -
- **array_keys().**
  - Muestra los índices o claves del array especificado.
  - En realidad, devuelve un array de claves indexado.

    - Se puede anidar a la función implode() para mostrar las claves con un separador intercalado.
    - Para comprobar su funcionamiento, hay que incluir esta función dentro de print_r().
    - Sintaxis:
        - Mostrar todas las claves o índices.
            - array_keys(nombre del array);
        - Mostrar la clave asociada a un valor especificado.
            - array_keys(nombre del array,"valor");
    - Ejemplo:
        - print_r (arrays_keys($numeros)); // Muestra todas las claves del array $numeros en una línea junto con su índice interno.
        - print_r (arrays_keys($numeros,24)); // Muestra el índice o la clave del array $numeros asociada al valor 24.
        - print_r (implode ("
", array_keys($numeros))); // Muestra cada clave del array $numeros en una línea distinta.
- **array_values().**
    - Muestra los valores o datos del array especificado.
    - En realidad, devuelve un array de valores indexado.
    - Se puede anidar a la función implode() para mostrar los valores con un separador intercalado.
    - Para comprobar su funcionamiento, hay que incluir esta función dentro de print_r().
    - Sintaxis:
        - array_values(nombre del array);
    - Ejemplo:
        - print_r (arrays_values($numeros)); // Muestra todos los valores del array $numeros en una línea junto con su índice interno.
        - print_r (implode ("
", array_values($numeros))); // Muestra cada valor del array $numeros en una línea distinta, al usar como separador un salto de línea.
- **array_combine().**
    - Crea un nuevo array asociativo combinando el contenido de otros 2 indexados.
    - El contenido del primero se usará como claves del combinado.
    - El contenido del segundo array como valores.
    - Sintaxis:
        - array_combine(nombre del array para claves, nombre del array para valores);
    - Ejemplo:
        - $a = array("nombre","apellidos","teléfono");
        - $b = array("Juan","López Fernández","123456789");
        - $c= array_combine($a,$b);
        - print_r($c);
- **[a][r] o [k][r]sort().**
    - Ordenar arrays con diferentes tipos de datos puede no hacerse bien.
    - El array ordenado queda modificado.

- Diferentes funciones de ordenación según tipo de array:
  - **sort().**
    - Ordena un array indexado de forma ascendente.
  - **rsort().**
    - Ordena un array indexado de forma descendente.
  - **asort().**
    - Ordena un array asociativo de forma ascendente.
  - **arsort().**
    - Ordena un array asociativo de forma descendente.
  - **ksort().**
    - Ordena las claves de un array asociativo de forma ascendente.
  - **krsort().**
    - Ordena las claves de un array asociativo de forma descendente.
  -
- Sintaxis:
  - [a][r]sort o [k][r]sort($array a ordenar, tipo de ordenación);
- Tipos de ordenación:
  - **SORT_REGULAR.**
    - Valor predeterminando.
  - **SORT_NUMERIC.**
    - Ordenación numérica.
  - **SORT_STRING.**
    - Ordenación alfabética.
  - **SORT_NATURAL.**
    - Ordena cadenas con números teniendo en cuenta el orden natural de éstos y no su orden alfabético, que coloca 11 antes que 8.
- Ejemplos:
  - $numeros = array(12,4,9,2,5,7);
  - $nombres = array(“primero”=>Juan”,segundo=> “Ana”, ”tercero”=> “Luis”);
  - sort($numeros); // resultado: 2, 4, 5, 7, 9,12.
  - rsort($numeros); // resultado: 12, 9, 7, 5, 4, 2.
  - sort($numeros, SORT_STRING); // resultado: 12,2,4,5,7,9.
  - arsort($nombres); // resultado: Luis, Juan, Ana.
  - ksort($nombres); // resultado: primero, segundo, tercero.
  - krsort($nombres); // resultado: tercero, segundo, primero.

- **max().**
  - Muestra el máximo valor contenido en un array.
  - Los números como cadenas de caracteres son convertidos a números reales o enteros.
  - Válido para Arrays indexado y asociativos.
  - Sintaxis.

        - max ($nombre del array);
    - Ejemplo:
        - $numeros = array(4,9,2,5,7);
        - echo “Número mayor: “ . max ($numeros); // Muestra 9.
- **min().**
    - Muestra el mínimo valor contenido en un array.
    - Los números como cadenas de caracteres son convertidos a números reales o enteros.
    - Válido para Arrays indexado y asociativos.
    - Sintaxis.
        - min ($nombre del array);
    - Ejemplo:
        - $numeros = array(4,9,2,5,7);
        - echo “Número mayor: “ . min ($numeros); // Muestra 2.
- **array_reverse().**
    - Invierte el contenido de un array.
    - Se pueden también invertir o no los índices.
    - Sintaxis:
        - array_reverse($nombre del array, preservar índice));
        - Preservar índice:
            - valor booleano.
                - ***true***:
                    - Se mantiene el índice original.
                - ***false***:
                    - Se invierten los índices.
                    - Valor por defecto.
            - Sólo funciona con índices numéricos, no con claves asociativas.
    - Ejemplos:
        - $numeros = array(4,9,2,5,7);
        - print_r(array_reverse($numeros));
        - print_r(array_reverse($numeros,true));
- **array_push().**
    - Inserta uno o más elementos al final de un array.
    - En un array asociativo los elementos insertan con índices, no con claves asociativas.
    - Para insertar uno o varios elementos al final del array asociativo, se incluyen con una inserción normal: $nombre array[“clave”] = valor;
    - Sintaxis:
        - array_push($nombre array, elemento 1 a insertar, elemento2 insertar, …);
    - Ejemplos:
        - $numeros = array(4,9,2,5,7);
        - array_push($numeros,10);
        - print_r($numeros); // Muestra 4,9,2,5,7,10.
        - array_push($numeros,10,80,100);
        - print_r($numeros); // Muestra 4,9,2,5,7,10,80,100.

- **array_pop().**
  - Elimina el último elemento de un array indexado o asociativo.
  - Sintaxis:
    - array_pop($nombre array);
  - Ejemplos:
    - $numeros = array(4,9,2,5,7);
    - array_pop($numeros);
    - print_r($numeros); // Muestra 4,9,2,5.
- **array_unshift().**
  - Inserta uno o más elementos al principio de un array.
  - En un array asociativo los elementos insertan con índices, no con claves asociativas.
  - Sintaxis:
    - array_unshift($nombre array, elemento 1 a insertar, elemento 2 insertar, …);
  - Ejemplos:
    - $numeros = array(4,9,2,5,7);
    - array_ unshift ($numeros,10);
    - print_r($numeros); // Muestra 10,4,9,2,5,7.
    - array_ unshift ($numeros,10,80,100);
    - print_r($numeros); // Muestra 10,80,100,4,9,2,5,7.
- **array_shift().**
  - Elimina el primer elemento de un array indexado o asociativo.
  - Sintaxis:
    - array_shift($nombre array);
  - Ejemplos:
    - $numeros = array(4,9,2,5,7);
    - array_shift($numeros);
    - print_r($numeros); // Muestra 9,2,5,7,10.
- **array_slice().**
  - Muestra una parte de un array.
  - Sintaxis.
    - array_slice($nombre array, índice, longitud, preservar índices).
    - Índice:
      - **Valor positivo**:
        - Índice inclusive desde el principio del array a partir del cual se extrae.
      - **Valor negativo**:
        - Índice desde el final del array a partir del cual se extrae.
    - Longitud: (opcional)
      - Número de elementos a mostrar, sólo si están disponibles.
    - Preservar índice. (opcional).
      - valor booleano.
        - ***true***:
          - Se mantiene el índice original.
        - ***false***:
          - Asigna índices nuevos empezando por 0.

    - Valor por defecto.
  - Ejemplos:
    - $numeros = array(4,9,2,5,7);
    - $varios = array(14, ”hola”,1);
    - print_r (array_ slice ($numeros, 2)); // Muestra 2,5,7.
    - print_r (array_ slice ($numeros, -2)); // Muestra 5 y 7.
    - print_r (array_ slice ($numeros, 2, 2)); // Muestra 2 y 5.
    - print_r (array_ slice ($numeros, 2, 2, true)); //Muestra los valores 2 y 5 con sus índices originales, [2] y[3] respectivamente.

- **array_splice().**
  - Permite insertar, modificar o eliminar elementos de un array en cualquier posición.
  - Sintaxis.
    - array_splice($nombre array, índice, longitud, array con valores a insertar).
    - Índice:
      - **Valor positivo**:
        - Índice inclusive desde el principio del array a partir del cual se eliminan elementos.
        - Muestra los elementos eliminados.
      - **Valor negativo**:
        - Elimina tanto elementos desde el final de array como se especifique.
        - Muestra los elementos eliminados.
    - Longitud. (opcional).
      - Número de elementos a eliminar, sólo si están disponibles.
      - Si no se especifica, se eliminan todos los elementos desde la posición especificada hasta el final.
      - Si se incluye un 0 y un array de reemplazo, no se eliminan elementos, sino que se insertan los nuevos en las posiciones especificadas y los elementos que las ocupaban quedan desplazados.
    - Array a insertar. (opcional).
      - Especifica el array con los elementos que sustituirán a los eliminados.
  - Ejemplos:
    - $numeros = array(4,9,2,5,7);
    - $numeros2 = array(20,30,40,50);
    - print_r (array_ splice ($numeros, 2)); // Elimina y muestra 2,5,7.
    - print_r (array_ splice ($numeros, -2)); // Elimina y muestra 5 y 7.
    - print_r (array_ splice ($numeros, 2, 2)); // Elimina y muestra 2 y 5.
    - print_r (array_ splice ($numeros, 2, 2, $numeros2)); //Sustituye los valores 2 y 5 por los del array numeros2, 4, 9, 20, 30, 40, 50 y 7.

        - print_r (array_ splice ($numeros, 2, 0, $numeros2)); //Inserción sin eliminación. El array números queda: 4, 9, 20, 30, 40, 50, 2, 5, y 7.

- **array_sum().**
    - Suma todos los valores numéricos de un array.
    - Si el array está vacío o contiene sólo cadenas de caracteres devuelve 0.
    - Si tiene una mezcla de números y cadenas de caracteres, suma los números.
    - Sintaxis:
        - array_sum($nombre array);
    - Ejemplo:
        - $numeros = array(4,9,2,5,7);
        - $varios = array(14, ”hola”,1);
        - print_r (array_ sum ($numeros)); // Devuelve 27.
        - print_r (array_ sum ($varios)); // Devuelve 15.
- **array_product().**
    - Multiplica todos los valores numéricos de un array.
    - Si el array está vacío o contiene sólo cadenas de caracteres devuelve 0.
    - Si tiene una mezcla de números y cadenas de caracteres, devuelve 0.
    - Sintaxis:
        - array_product($nombre array);
    - Ejemplo:
        - $numeros = array(4,9,2,5,7);
        - $varios = array(14, ”hola”,1);
        - print_r (array_ product ($numeros)); // Devuelve 2520.
        - print_r (array_ product ($varios)); // Devuelve 0.
- **array_diff().**
    - Permite comparar arrays entre sí para comprobar si tienen los mismos elementos o no.
    - Funcionamiento:
    - Compara el primer array que se incluya como argumento en la función con otros arrays, y muestra los valores del primero que no están en el segundo.
    - Si el primer array no tiene elementos distintos, devuelve Array().
    - Si se comparan 2 arrays usando cada uno en primer lugar en cada comparación se podría, comprobar si son iguales.
    - Sintaxis:
        - array_diff($nombre array 1, $nombre array 2, …, $nombre array N);
    - Ejemplos:
        - $numeros1 = array (4,9,2,5,7);
        - $numeros2 = array (14, 23,6,2,5,7);
        - $numeros3 = array (4,9,2,5,7,8);
        - print_r (array_ diff ($numeros1, $numeros2)); // Devuelve Array([0]=>4, [1]=>9).
        - print_r (array_ diff ($numeros1, $numeros3)); // Devuelve Array(), es decir, que los elementos del primer array están incluidos en el segundo.

FUNCIONES.

Concepto.

- Conjunto de instrucciones u operaciones agrupadas dentro de un mismo bloque.
- Ahorrar escribir código que se repite con frecuencia.
- Pueden ejecutarse en cualquier parte de un programa.
- Pueden devolver o no valores.
- Las funciones son independientes del resto del programa. Las variables de una función son independientes de las variables del programa principal. Inicialmente, ni la función tiene acceso a las variables del programa principal (a no ser que se use global), ni el programa principal tiene acceso a las variables de la función (por ser locales).
- Las funciones tienen ámbito global, es decir, se pueden llamar desde otra función incluso si se han creado dentro de otras funciones.
- Funciones miembro:
    - Funciones que forman parte o son miembros de una clase.
    - Dentro de una clase se denominan métodos.

Tipos.

- Creadas por el usuario.
- Internas:
    - Incluidas en PHP.

FUNCIONES CREADAS POR EL USUARIO.

- **Creación o definición de una función.**
    - Se usa la palabra reservada *function***.**
    - Pueden declararse en cualquier parte del código.
    - Pueden o no incluir argumentos**.**
    - Sintaxis básica:

```
function nombre_función(argumento 1, argumento 2,….,argumento N)
{
        Instrucciones;
}
```

    - Nombre de una función:
        - Se puede usar caracteres alfanuméricos (letras y números) y guiones de subrayado.
        - Insensible a mayúsculas y minúsculas. (suma (), SUMA () o SumA () son la misma función).
        - No se pueden usar nombres de funciones ya creadas o palabras reservadas.
        - Se puede usar nomenclatura camelCase, aunque es muy común usar guiones bajos para separar palabras y escribir todo el nombre de la función en minúscula:
            - calcular_area();

- Bloque de código:
  - Se encierra entre llaves {}.
  - Puede incluir:
    - Declaración de variables y constantes.
    - Bucles e instrucciones condicionales.
    - Llamadas a otras funciones.
    - Etc.
- Argumentos:
  - Son opcionales y se denominan parámetros.
  - Datos que recibe la función y que permiten realizar las operaciones dentro de la misma.
  - Indican a una función los valores con los que tiene que operar.
  - Si no se incluyen, la función utiliza los valores fijos que se especifican en su bloque de instrucciones, y siempre hace lo mismo.
  - Se especifican dentro de los paréntesis de la función.
  - Son valores que se pasan a la función en el momento de su ejecución.
  - Se especifican como variables dentro de los paréntesis de la función.
  - Si son varios, se separan por una coma.
  - Se igualan a las variables de la clase o función para así asignarlas un valor.
  - Pueden pasarse por valor o por referencia.
- Valores como parámetros:
  - Valores literales, variables, expresiones aritméticas, funciones anidadas, objetos, etc.,
  - El parámetro se inicializa con dichos valores.
- Paso de argumentos a una función.
  - **Por valor**:
    - Forma más común.
    - Se pasa el valor contenido en una variable.
    - Una vez pasados los argumentos se les puede cambiar su valor dentro de la función, sin embargo, las variables que se han utilizado, fuera de la función, seguirán teniendo su valor original.
    - Ejemplo:

```
function suma_dos($n1, $n2)
{
        $suma = $n1 + $n2;
        return $suma;
}
```

  - **Por referencia**.
    - Si tras pasar los argumentos se les cambia su valor dentro de la función, los valores de las variables que se han utilizado también se modifican fuera de la función.
    - Ejemplo:

```
function suma_dos(&$n1, &$n2)
{
        $n1 = 20;
        $n2 = 40;
}
```

- **Por valor predeterminado.**
  - Valores por defecto que se incluyen al definir el parámetro para ser utilizados si se llama a la función y no se incluyen todos o alguno de ellos.
  - Para evitar ambigüedades, deben ser los últimos en la lista de argumentos, es decir, sólo pueden ser opcionales aquellos parámetros situados a la derecha del último que sea necesario para realizar las acciones de la función.
  - Todos pueden ser opcionales.
  - Sintaxis:

```
function nombre de la función (parámetro 1 = valor, parámetro 2= valor, …, parámetro N)
{
        instrucciones;
}
```

  - Al llamar la función se pueden o no incluir los parámetros opcionales. Si se incluyen sus valores, son estos los que utiliza la función, si no, se utilizan los valores definidos como valores por defecto.
  - Se usan como parámetros los valores en el mismo orden en que se han incluido en la llamada a la función.

```
function calcular($n1=23, $n2=4,$n3=4, $n4=8)
{
    return $n1 + $n2 + $n3 + $n4;
}
$a = calcular (20,50,70,30);
$b = calcular (30,50); Equivale a calcular (30,50,4,8);
$c = calcular (); Equivale a calcular (23,4,4,8);
```

  - El orden de los valores es importante, porque si no se respeta pueden ocurrir errores:

```
function calcular ($n1=23, $n2, $n3=4, $n4)
{
    return $n1 + $n2 + $n3 + $n4;
}
$a = calcular (20,50,70,30);
$b = calcular (30,50); //Error, porque el cuarto parámetro se queda sin valor, al usar los valores en el mismo orden en el que se han especificado en la función.
```

- **Llamada a una función:**
  - Ejecución de una función.
  - Una función se puede ejecutar o llamar varias veces.
  - Desde una función se pueden ejecutar otras.
  - Al llamarla, todas las instrucciones que incluya en su bloque de código se ejecutan.
  - No se puede llamar a una función que no esté definida.
  - Sintaxis:
    - nombre de la función ();
  - Ejemplo:
    - calcular_saldo();

TIPOS DE FUNCIONES.

- **Sin retorno de datos y sin parámetros.**
  - Sintaxis**:**

```
function nombreFunción()
{
        Instrucciones;
}
```

    - Ejemplo:

```
function sumar()
{
        $suma = $n1 + $n2;
        echo $suma;
}
sumar();
```

- **Sin retorno de datos y con parámetros.**
  - Sintaxis:

```
function nombreFunción(nombre parámetro 1, nombre parámetro 2,…,nombre parámetro N)
{
        Instrucciones;
}
```

  - Ejemplo:

```
function sumar3($numero1, $numero2, $numero3)
{
        echo ("El resultado de la suma es: “. ($numero1+$numero2+$numero3) . "<br>");
}
suma3(5,8,9);
```

- **Con retorno de datos y sin parámetros.**

- Si la función produce un resultado y se quiere utilizar después, hay que almacenar éste en una variable, constante, …
- Para devolver el valor se utiliza la instrucción **return**.
- Sintaxis**:**

```
function nombreFunción()
{
    Instrucciones;
    return valor a devolver;
}
```

- Llamada a la función**:**
  - Hay que crear una variable y usar una operación de asignación para que se cargue en ella el valor devuelto.
  - Sintaxis:
    - $variable que almacenara el valor devuelto = función();
  - Ejemplo:

```
function restar()
{
    $a = 7;
    $b = 5;
    return a-b;
}
$resta = restar();
echo $resta;
```

- **Con retorno de datos y con parámetros.**
  - Si la función produce un resultado y se quiere utilizar después, hay que almacenar éste en una variable, constante, …
  - Para devolver el valor se utiliza la instrucción **return**.
  - Incluyen parámetros para pasar datos a la función.
  - Sintaxis**:**

```
function nombreFunción(nombre parámetro 1, nombre parámetro 2,…)
{
        instrucciones;
        return valor a devolver;
}
```

  - Llamada a la función**:**
    - Hay que crear una variable y usar una operación de asignación para que se cargue en ella el valor devuelto.
    - La función debe incluir obligatoriamente los valores para los parámetros con los que se ha definido.
    - Sintaxis:
      - var variable que almacenara el valor devuelto = función (valores para los parámetros);
    - Ejemplos:

```
function sumar3($a, $b, $c)
{
    $d = $a + $b + $c;
    return $d;
    //return $a + $b + $c;
}
```

Opción con datos literales:

```
    $resultado1 = sumar3(7,9,5);
```

Opción con datos dentro de variables:

```
    $a1 = 7;
    $a2 = 9;
    $a3 = 5;
    $resultado2 = sumar3($a1, $b1, $c1);
```

Retorno de varios datos.

- **Con Arrays.**
  - En una función que devuelve datos cuando es ejecutada no se puede usar más de una instrucción return.
  - Si se quisiera devolver más de un resultado calculado dentro una función, estos resultados deben incluirse en un array.
  - Después cada valor del array devuelto se incluye en una variable diferente usando la función list().
  - Sintaxis:

```
function función()
{
        return array($resultado 1, $resultado 2, … $resultado N];
}
list($variable1, $variable2, …, $variable N) = función ();
```

  - Ejemplo:

```
function calcular($n1, $n2)
{
        $suma = $n1 + $n2;
        $resta = $n1- $n2;
        return array($suma, $resta);
}

 list($num1, $num2) = calcular (5,8);
```

- **Con return alternativos.**

- En este caso, solo se devuelve un resultado con return, pero puede haber varios de ellos que se ejecutan según se cumplan unas u otras condiciones.
- Ejemplo:

```
function comparar(($n1, $n2)
{
        If (($n1> $n2)
        {
                return “n1 es mayor”;
        }
        else If (($n1< $n2)
        {
                return “n2 es mayor”;
        }
        else If (($n1== $n2)
        {
                return $n1 + $n2;
        }
}
```

Funciones anónimas.

- También se las denomina cierres o closures.
- Son obligatorias cuando hay que pasar una función como un parámetro de otra.
- También, se puede definir y almacenar el bloque de instrucciones dentro una variable y después ejecutarlo.
- El nombre de la variable será el nombre de la función, ya que la función no tiene nombre.
- Las funciones creadas así tienen las mismas características que las creadas de otras formas (pueden tener o no parámetros y pueden o no retornar datos.).
- Variables del ámbito padre pueden ser usadas por valor o por referencia usando la cláusula use.
- Sintaxis:

```
$nombre = function (Con o sin parámetros)
{
        instrucciones;
}
```

- Ejemplos:

```
$sumar = function (n1, n2)
{
        return n1 + n2;
}
$restar = function ()
{
        $n1 = 4;
```

```
        $n2 = 2;
        $n3 = $n1 – $n2;
        echo “Resultado resta: “ . $n3;
    }
```

- Ejecución o llamada de una función anónima:
  - Igual que cualquier otra función.
  - Sintaxis:
    - nombre de la función (Con o sin parámetros);
    - $variable = nombre de la función (Con o sin parámetros);
  - Ejemplos:

```
        restar();
        $resultado = sumar(4,2);
        echo “Resultado suma: “  . $resultado;
```

Funciones flecha (arrow).

- Cierres cortos o short closures.
- Forma abreviada de definir una función.
- Se elimina la palabra function y se sustituye por fn.
- Desaparece la cláusula return.
- Sólo contienen una expresión, que es el retorno de la función.
- Sólo pueden contener una línea de código.
- No se pueden usar las llaves {}.
- Variables del ámbito padre pueden ser usadas por valor automáticamente.
- Sintaxis de declaración:

```
fn (parámetros)  => expresión;
```

- Ejemplos:

```
$total = fn  ($cantidad, $precio) => $cantidad * $precio;
```

Equivale a:

```
$total = function ($cantidad, $precio)
{
        return $cantidad * $precio;
}
```

- Ejecución o llamada de una función flecha:
  - Igual que cualquier otra función.
  - Sintaxis:
    - nombre de la función (Con o sin parámetros);
    - $variable = nombre de la función (Con o sin parámetros);
  - Ejemplos:

```
$total = sumar1(4,2);
echo “Total: “ . $total;
```

CLASES.

- **Concepto.**
  - Plantilla o modelo que define a los objetos.
  - Incluyen las características (propiedades) y comportamiento (métodos), que definen a un tipo concreto de objetos.
  - Es decir, una clase encapsula variables y funciones o métodos.
- **Creación.**
  - Las propiedades o atributos de una clase se crean usando variables.
  - Las variables pueden tener valores iniciales.
  - Los métodos se implementan mediante funciones.
  - **class**.
    - Palabra clave o instrucción que permite definir una clase.
  - Sintaxis:

```
class nombre de la clase {
        [alcance] $variable1; // Propiedad de la clase usando variables.
        [alcance] $variable2;
        [alcance] $variableN;
        [alcance]  function nombre método 1([$parámetro 1, $parametro2,…,parámetro N])
        {
                Instrucciones;
                [return;]
        } //Métodos que indican cual es el comportamiento de los objetos.
        [alcance]  function nombre método 2([$parámetro 1, $parametro2,…,parámetro N])
        {
                Instrucciones;
                [return;]
        }
        [alcance]  function nombre método N([$parámetro 1, $parametro2,…,parámetro N])
        {
                Instrucciones;
                [return;]
        }
}
```

- **Variables o atributos**
  - Definen las propiedades de la clase.
  - Sintaxis:
    - [alcance] $nombre de la variable;
    - Se crean en el interior de las clases como cualquier otra variable.
    - Puede usarse la cláusula var antigua.
  - Tipos:
    - Las que toman valores distintos para cada objeto.
    - **Estáticas**:

- También se las denomina variables de clase.
- Toman valores iguales para todos los objetos.
- Todos los objetos que comparten la variable tendrán el mismo valor en ella.
- Si se modifica el valor, este se modifica en todos los objetos de la clase.
- Uso de la instrucción o cláusula static.
- A una propiedad estática se puede acceder sin necesidad de crear objetos o instanciar clases.
- Para acceder a ellas no se usa el operador flecha (->), si no, el operador de resolución de alcance (::).
- Para acceder a ellas dentro de las clases no se usa la pseudovariable $this, sino self:: .
- Sintaxis:
  - static $nombre de la variable = valor;
- Ejemplo:
  - static $num_ruedas = 4;

- **Métodos.**
  - Se crean a través de funciones (function).
  - Pueden o no incluir parámetros.
  - Los métodos de clase o estáticos no necesitan que la clase que los contiene sea instanciada, se puede acceder a ellos in usar ningún objeto.
  - Parámetro:
    - Valores que se pasan a la función en el momento de su ejecución.
    - Se especifican como variables dentro de los paréntesis de la función.
    - Si son varios, se separan por una coma.
    - Se igualan a las variables de la clase o función para así asignarlas un valor.
  - Sintaxis:
    - [alcance] function nombre método ([$parámetro 1, $parametro2,…,parámetro N])
  - Ejemplos:
    - **Con parámetros y devuelve datos**:

```
public function areaCirculo($radio)
{
    $area = M_PI * ($radio**2);
    return $area;
}
```

    - **Con parámetros y no devuelve datos**:

```
public function mover($x, $y)
{
    $posx = $x;
    $posy = $y;
}
```

- **Sin parámetros y devuelve datos**:

```
public function sumar($radio)
{
    $suma = 7+2;
    return $suma;
}
```

- **Sin parámetros y no devuelve datos**:

```
public function imprimir()
{
    echo “Hola Mundo”;
}
```

- **Métodos estáticos.**
  - Las propiedades y métodos estáticos con elementos que pertenecen a la clase y no a los objetos, es decir, son compartidos por todas las instancias.
  - Método accesible sin necesidad de crear un objeto o instanciar una clase.
  - Se accede al método a través del operador de resolución de alcance (::), si bien, también se puede acceder a él a través de un objeto.
  - Útil cuando se quiere mantener el valor devuelto por la ejecución de un método para usarlo en otras operaciones.
  - Estos métodos no pueden acceder a las propiedades no estáticas de la clase, ya que no tienen un objeto de ésta.
  - Si pueden acceder a otras propiedades y métodos estáticos de la clase.
  - Sintaxis:

```
[alcance] static function Nombre del método ()
{
Instrucciones;
};
```

  - Ejemplo:

```
class Persona
{
    static $nombre; // propiedad estática;
    static fuction mostrarDatos()
    {
        echo $this->nombre; // Error, no se puede acceder a propiedades no estáticas en el interior de un método estático.
        echo $this::nombre // Correcto, acceso a propiedad estática.
    }
}
Persona::mostrarDatos; // Acceso a método a través de la clase.
```

- **Métodos constructores.**
  - Método especial que permite crear objetos de una clase.
  - Permiten inicializar un objeto antes de ser utilizado.

- No se incluyen en ellos variables estáticas.
- Uso no obligatorio.
- Sólo puede haber uno por clase.
- Nombre del método:
  - __construct()
- Sintaxis:
  - **Sin parámetros**:

```
function __construct()
{
        instrucciones;
}
```

  - **Con parámetros**: (Variables y parámetros pueden o no tener el mismo nombre).

```
function __construct($parámetro1, $parámetro2,…,$parámetroN)
{
        $this->variable1 = $parametro1;
        $this->variable2 = $parametro2;
        $this->variableN = $parametroN;
}
```

- Ejemplos:
  - **Sin parámetros**:

```
function __construct()
{
        echo “Cualquier frase”;
}
```

  - **Con parámetros**:

```
private $nombre;
private $apellidos;
private $edad;
function __construct($a, $b, $c )
{
        $this->nombre = $a;
        $this->apellidos = $b;
        $this->edad=$c;
}

$persona1 = new Persona(“Juan”, “López Martín”, 30); // Creación de un objeto.

$edad = 20;
$persona1->edad = 20;
$persona2->edad = 25;
```

- **$this.**

- Palabra reservada que hace referencia al propio objeto que se está creando, es decir, hace referencia al objeto actual.
- No se puede hacer referencia a métodos estáticos usando $this, pero si a métodos públicos, privados y protegidos.
- Variable o pseudovariable que permite:
  - Distinguir las variables miembros de los parámetros.
  - Acceder a alguna propiedad o método de un objeto en la propia definición de la clase.
- Sintaxis:

```
$this->nombre variable = $parametro;
```

- Ejemplo:

```
$this->edad = $numero;
```

$numero es un parámetro incluido dentro de una función o método de la definición de la clase.

- **self.**
  - Hace referencia a la clase actual.
  - Se usa cuando no se instancia dicha clase, es decir, cuando se utilizan constantes y métodos de clase o estáticos.
  - Sintaxis:

```
self::CONSTANTE;
self::método estático();
```

  - Ejemplo:

```
const IVA;
public static function método_estatico(){ …}
echo “El valor del IVA es: “ . self::IVA;
echo self::método_estatico();
```

- **Métodos destructores.**
  - Un objeto creado existe mientras se ejecuta el código PHP y finaliza, o se elimina, tras acabar la ejecución.
  - Aunque un objeto se destruye solo, puede ser necesario en ocasiones hacerlo de forma explícita para cerrar recursos como archivos, conexiones a base de datos, etc.
  - También se puede destruir un objeto en un momento dado sin esperar a que finalice su ejecución usando el método unset (nombre del objeto).
  - Sintaxis:

```
public function __destruct()
{
    echo “mensaje”;
}
```

    - Mensaje:
      - Un simple mensaje permite comprobar que se está ejecutando el código destructor.

- **Métodos getters and setters.**

- Como las propiedades suelen definirse con alcance privado, no se puede acceder a ellas desde fuera de las clases.
- Los métodos de una clase suelen ser públicos para poder acceder a ellos desde el exterior.
- Se puede, por tanto, acceder indirectamente a propiedades privadas a través de los métodos públicos definidos en la clase.

Métodos (empiezan por):

- **get.**
  - Permite obtener el valor de una propiedad protegida o privada.
  - El método empieza por get (aunque no es obligatorio).
  - Suele ponerse get y el nombre de la propiedad a la que se accede.
  - No lleva parámetros.
  - Sintaxis:

```
public function getNombreADar()
{
        return $this->variable con alcance privado o protegido;
}
```

  - Ejemplo:

```
class Gente
{
private $nombre;
private $edad;
function getNombre()
{
        return $this->nombre
}
function getEdad()
{
        return $this->edad;
}
```

  - Uso:
    - A través de un objeto.
  - Sintaxis:

```
objeto->getMetodo();
```

  - Ejemplo:

```
$edadPersona = persona1->getEdad(); //Suponiendo un objeto llamado persona1.

echo “Te llamas: “ . persona1->getNombre();
```

- **set.**
  - Permite asignar o modificar el valor de una propiedad protegida o privada.
  - Las propiedades deben protegerse del acceso exterior para evitar que se introduzcan datos erróneos, haciendo la entrada de datos

desde un método accesorio (setMetodo()), que será público, siendo la propiedad privada o protegida.

- El método empieza por set (aunque no es obligatorio).
- Suele ponerse set y el nombre de la propiedad a la que se accede.
- Incluye un parámetro que será el valor que dar a la propiedad.
- Puede también incluir el nombre de la propiedad como parámetro.
- No tiene sentido un método set si la propiedad es de solo lectura y no se puede modificar.
- **Modificador readonly (de solo lectura)**:
  - Impide la modificación de la propiedad después de la inicialización.
  - Sintaxis:
    - alcance readonly $propiedad;
  - Ejemplo:
    - private readonly $saldo;

- Métodos set.
  - Sintaxis:

```
public function setNombreADar($parametro)
{
        $this->variable = $parametro;
}
```

  - Ejemplo:

```
Class Gente
{
private $nombre;
private $edad;
private $ciudad;
function setCiudad($nombre_ciudad)
{
        $this->ciudad = $nombre_ciudad ;
}
```

  - Uso:
    - A través de un objeto.
  - Sintaxis:

```
objeto->setMetodo(valor del parámetro);
```

  - Ejemplo:

```
persona1->setCiudad("Teruel"); //Suponiendo un objeto llamado persona1.
```

- Otras sintaxis:

```
public function  set($parámetro 1, $parámetro 2)
{
        $this->variable(El parámetro 1) = $parametro2;
}
```

  - $parámetro 1: Es la variable a la que se le asigna valor.

- $parámetro 2: Es el valor.

```
function public set($parámetro 1, $parámetro 2)
{
        If(property_exists($this, $nombre variable)
        {
        $this->variable(El parámetro 1) = $parametro2;
        }
}
```

- Ejemplo:

```
function set($ciudad, $nombre_ciudad)
{
        $this->ciudad = $nombre_ciudad ;
   }
persona1->set($ciudad, "Teruel");

function public set($ciudad , $ciu)
{
        If(property_exists($this, $ciudad)
        {
                $this->ciudad = $ciu;
        }
}
persona1->set($ciudad, "Toledo");
```

- **Visibilidad o acceso a los miembros de una clase.**
  - Para controlar el acceso a una propiedad o método de una clase se usan modificadores de alcance.
  - Son obligatorios.
  - **Modificadores**:
    - **private**.
      - A una propiedad o método declarado como privado solo se puede acceder desde la clase donde está definido.
      - Una subclase que herede de una clase con métodos o propiedades privados no puede acceder a ellos.
      - Por seguridad las propiedades se definen como privadas o protegidas a no ser que puntualmente sean públicas o estáticas.
      - Solo se podrá acceder a las propiedades con métodos que interactúen con ellas.
      - Sintaxis:
        - private $variable = valor;
        - private function nombreMetodo(){instrucciones;}
      - Ejemplos:
        - private $dni ="012345678M";
        - private calcularSaldo(){…}
    - **public**.

        - Acceso sin restricciones a propiedades y métodos.
        - Si no se incluye, es el modificador por defecto para todas las propiedades y métodos. (En las propiedades hay que incluir var).
        - Sintaxis:
            - public $variable = valor;
            - public function nombreMetodo(){instrucciones;}
        - Ejemplos:
            - public $nombre = ”Lola”;
            - public imprimir(){…}
    - **protected**.
        - Menos restrictivo que private.
        - A una propiedad o método protegido se puede acceder desde la clase donde están definidos y desde subclases derivadas por herencia.
        - Los métodos que no necesiten acceso desde el exterior de la clase deben definirse como protegidos.
        - Sintaxis:
            - protected $variable = valor;
            - protected function nombreMetodo(){instrucciones;}
        - Ejemplos:
            - protected $edad =30;
            - protected function getEdad() {return $this->edad;}
- **Ubicación de las clases y organización del código.**
    - Las clases se suelen guardar en archivos independientes a los que hay que llamar o incluir en la página donde se vaya a utilizar.
    - Funciones de llamada:
        - **include().**
        - **require().**
        - **include_once() y require_once().**
            - Incluyen el archivo correspondiente una sola vez.
            - Así se evitan errores si se hacen varias llamadas para incluir el mismo archivo.
            - Tienen las mismas características de tolerancia a fallos que include() y require().
            - Pueden usarse sin paréntesis.
            - Sintaxis:
                - include_once(“ruta de acceso/archivo.php”);
                - require_once(“ruta de acceso/archivo.php”);
            - Ejemplos.
                - include_once(“archivo1.php”);
                - require_once(“misitio/PHP/archivo2.php”);

OBJETOS.

- **Creación**.
    - Los objetos o instancia de clase se crean a partir de la clase.
    - Incorporan valores concretos para las propiedades de la clase.
    - Sintaxis:

        - **Constructor sin parámetros**.
            - $nombre del objeto = new Nombre Clase ();
        - **Constructor con parámetros**.
            - $nombre del objeto = new Nombre Clase (Valor parámetro 1, valor parámetro 2, …, valor parámetro N);
    - Ejemplos:
        - $persona1 = new Persona ();
        - $persona2 = new Persona ("Lily", "James", "C/Sevilla, 60");
- **Acceso a miembros del objeto.**
    - Sintaxis:
        - $nombre del objeto->propiedad;
        - $nombre del objeto->metodo();
    - Ejemplos:
        - $persona1->$nombre; // Si $nombre es una propiedad pública.
        - $persona2->getDireccion (); // Si $direccion es una propiedad privada.
        - $persona1->$edad = "34"; Si $edad es una propiedad pública.
        - $persona2-setTelefono ("987654321"); //Si $telefono es una propiedad privada.
        - $persona1->calcular_año_nacimiento ();
        - $persona2->calcular_año_nacimiento (34);

HERENCIA.

- Se pueden crear clases a partir de otras clases mediante el mecanismo de la herencia, los que permite el ahorro de código.
- Una relación de herencia es del tipo "es un", por ejemplo, un vendedor es un empleado. (De subclase a superclase y no al revés).
- **Tipos de clases en herencia**.
    - Clase base, padre o superclase:
        - Es la clase más genérica y de la que se hereda.
    - Clase derivada, hija o subclase:
        - Es la clase más específica y es la que hereda.
- **Características de la subclase**.
    - Obtiene todas las variables miembro de la superclase.
    - Obtiene todos los métodos miembro de la superclase.
    - Puede definir nuevas variables y métodos o funciones.
    - Sintaxis.
        - class subclase extends superclase{…}
    - Ejemplo:

```
class Empleado
{
        protected $nombre;
        $antiguedad;
        $sueldo;
        $hora_entrada;
        $hora-salida;
        $diario;
public function horas-diarias ()
```

```
{
        $this->diario = $this->hora_salida - $this->hora_entrada;
}

class Vendedor //(subclase) extends Empleado //(superclase)
 {
        private $ventas;
        private $comision;
public function vender($cantidad)
{
        $this->ventas = $this->ventas +$cantidad;
}
class Administrativo extends Empleado {
        private $horas_extras;

public function prima()
{
         echo "Prima por beneficios: " . $prima;
}
```

Creación de objetos en una relación de herencia.

- El objeto se crea llamado a los constructores de la subclase y de superclase.
- No es obligatorio crear constructor de la subclase, PHP llamará al de superclase.
- Se llama indirectamente al de la subclase y ésta llama al de la superclase.
- Para ello se usa la instrucción "parent" en el constructor de la subclase.
- Sintaxis.

```
public function __construct ($parámetro1, $parámetro2, $parámetro 3, …)
 {
        parent::__construct($parámetro1, $parámetro2, $parámetro 3, …);
        Instrucciones;
}
```

- Ejemplo:

```
class Vendedor extends Empleado
{
        public function __construct ($nombre, $sueldo, $fecha, $comision) {
        parent::__construct($nombre, $sueldo, $fecha); // Heredados de la superclase.
        $this->ventas = 0;
        $this->comisión = $comision; // Parámetro y variable de la subclase.
}
```

SOBRECARGA DE MÉTODOS.

- En PHP no es posible la sobrecarga de métodos.
- **Concepto**.
  - Característica de la POO que consiste en una nueva implementación de un método en una clase hija respecto del mismo método en la clase padre.
  - Un objeto puede contener varios métodos que se llamen igual, pero que difieran en su número de argumentos.
- **Alternativas para simular la sobrecarga de métodos en PHP.**
  - **Valores por defecto.**
    - Se incluyen valores por defecto como parámetros o argumentos de la función.
    - Sintaxis:
      - parámetro = null;
    - Ejemplo:

```
function Sumar($a1, $b1=null)
{
        if($b1 == null)
        {
                return $a1
        }
        else
        {
                $suma = $a1+$b1;
                return $suma;
        }
}
$a = sumar(4); // Resultado 4.
$b = sumar(3,5); // Resultado 8.
```

  - **Función func_get_args().**
    - Devuelve un array con todos los argumentos de una función.
    - No tiene parámetros.
    - Sintaxis:
      - func_get_args().
    - Ejemplo:

```
function hello()
{
        $argumentos = func_get_args();
        $numeroDeArgumentos = count($argumentos);

        if ($numeroDeArgumentos == 0)
{
            echo “Hola”<br>;
}
        else if ($numeroArgs == 1)
```

```
        {
            echo “Hola “ .$argumentos[0].</br>;
            }
    }
        hello();

hello(“Mundo.”);
```

CLASES Y MÉTODOS ABSTRACTOS.

- **Clase abstracta.**
  - Es una clase a partir de la cual no se pueden crear objetos, es decir, es una clase que no se puede instanciar.
  - Se utiliza como base o plantilla para crear otras clases.
  - Es, con las clases hijas o derivadas de una abstracta, con las que ya se pueden crear objetos.
  - Una clase abstracta incluye al menos un método abstracto.
- **Método abstracto**.
  - Es un método sin implementar.
  - Su implementación debe hacerse en las clases hijas que heredan de la abstracta.
  - Estos métodos deben ser definidos con la misma visibilidad, o con una menos restrictiva, que la indicada en la clase abstracta y tener el mismo número de argumentos.
- Utilidad:
  - Reducen la cantidad de código duplicado y mejoran su calidad**.**
- **abstract:**
  - Palabra clave o instrucción que permite definir una clase o un método abstracto.
- Sintaxis:
  - **Clases**:

```
abstract class nombre de la clase abstracta
{
Propiedades y métodos;
}
```

  - **Métodos**:
    - abstract visibilidad function nombre del método (Con o sin parámetros);
    - Visibilidad:
      - public.
      - private.
      - protected.
- Ejemplos:
  - **Clases y métodos abstractos**:

```
abstract class ClaseAbstracta
{
   public function getEdad($edad)
```

```
    {
        return $edad;
    } // Método definido.
    abstract public function setNombre($nombre); // Método abstracto.
    abstract public function getNombre(); // Método abstracto.
    }
```

- **Clases hija que hereda de clase abstracta:**

```
class MiClase extends ClaseAbstracta
{
    private $nombre;
    Otras propiedades y métodos;

    public function setNombre($nombre)
    {
        $this->nombre = $nombre;
    } // Implementación de método abstracto clase hija.

    public function getNombre()
    {
        return $this->nombre;
    } // Implementación de método abstracto clase hija.
}
```

Creación un objeto de una clase hija que hereda de una clase abstracta.

```
$cliente1 = new MiClase();
$nombre = $cliente1->getNombre();
echo "Nombre del cliente: " . $nombre . "<br>”;
```

CLASES FINALES.

- **Clase final.**
  - Clase de la cual no se puede heredar, es decir, no puede tener clase hijas.
  - Una clase puede ser declarada final si su definición está completa y no se necesitan subclases de ella.
  - Si que quiere heredar de una clase final, se produce un error.
  - Una razón para su uso es la seguridad. Así, se anula la posibilidad de crear clases derivadas que podrían tener código dañino, o código que anulase un método, propiedad o constante de la clase padre.
- **Método final**.
  - Es un método que no puede ser sobrescrito en una clase hija.
  - Un método privado no puede declararse como final desde la versión 8.0.0 de PHP, excepto el constructor.
- Las propiedades pueden declararse como finales a partir de la versión 8.4.0 de PHP.
- Las constantes a partir de la versión 8.1.0.
- **final:**

- Palabra clave o instrucción que permite definir clase, métodos, propiedades y constantes finales.

- Sintaxis:
  - **Clases**:

```
final class nombre de la clase abstracta
{
        Propiedades y métodos;
}
```

  - **Métodos**:
    - final visibilidad function nombre del método (Con o sin parámetros);
    - Visibilidad:
      - public.
      - private.
      - protected.
  - **Propiedades:**
    - final visibilidad $variable;
  - **Constantes:**
    - final visibilidad const Nombre de la constante = “valor”;

- Ejemplos:
  - **Clase Final**:

```
final class Cliente
    {
        public $nombre;
        public function Saludo()
            {
                print "Hola Cliente<br>";
            }
    }
class Cliente1 extends Cliente // Error. Cliente es una clase final.
    {
    }
```

  - **Clase con métodos, propiedades o constantes finales.**

```
class Padre
{
  final protected $variable; //A partir de versión 8.4.0.
  final public const CONSTANTE = "PHP"; //A partir de versión 8.1.0.
  final public function Saludo()
  {
      print "Hola Persona<br>";
  }
}
class Hija extends Padre
 {
   public $variable; // Error, $variable es final.
```

```
    public const CONSTANTE = "JavaScript"; // Error, CONSTANTE
    es final.
     public function Saludo()
    {
          echo "! Buenos Días ¡";
    } // Error, Saludo() es un método final.
}
```

INTERFACES.

- **Concepto.**
  - Una interfaz es una plantilla que define un conjunto de métodos que una clase debe implementar.
  - Así, una clase adquiere funcionalidad adicional, funcionalidad que está disponible para todas las clases que implementen una misma interfaz.
  - Todos los métodos declarados en una interfaz deben ser públicos.
- **Características.**
  - **Declaración de métodos abstractos.**
    - Define un conjunto de métodos, pero sin implementar.
    - Estos métodos, pueden tener o no parámetros.
    - Cualquier clase que utilice una interfaz debe implementar todos los métodos de ésta.
    - Si una clase no necesita un método determinado, se implementa obligatoriamente, pero se deja vacío.
  - **Herencia múltiple.**
    - Una clase puede implementar varias interfaces distintas.
  - **Polimorfismo.**
    - Una misma interfaz puede ser utilizada por distintas clases y cada una tener implementaciones distintas para los métodos que dicha interfaz tiene definidos.
  - **Constantes.**
    - No pueden contener propiedades, pero si constantes.
    - Éstas funcionan como las constantes de clase, pero no pueden ser sobrescritas por la clase que las herede o implemente.
  - **Interfaz extendida.**
    - Una interfaz puede heredar de otra interfaz.
    - Así, varias interfaces se pueden reutilizar y combinar para aumentar su funcionalidad.
    - También se denominan interfaces extensibles.
- **Creación de un interfaz**.
  - Para ello se utiliza la palabra reservada *interface.*
  - Sintaxis:

```
interface Nombre de la Interfaz

{
```

```
        public function nombre método 1 (con o sin parámetros);

        public function nombre método 2 (con o sin parámetros);

        public function nombre método N (con o sin parámetros);

    }
```

- **Creación de una interfaz extendida / extensible**.
    - Para ello se utiliza la palabra reservada *extends,* como cuando se hereda de una clase.
    - Sintaxis:

```
    interface Nombre de la Interfaz1

    {

        public function nombre método 1 (con o sin parámetros);

        public function nombre método 2 (con o sin parámetros);

        public function nombre método N (con o sin parámetros);

    }


    interface Nombre de la Interfaz2 extends Interfaz1

    {

        public function nombre método 1 (con o sin parámetros);

        public function nombre método 2 (con o sin parámetros);

        public function nombre método N (con o sin parámetros);

    }
```

- **Uso de una interfaz.**
    - Al definir una clase que implementa una interfaz se utiliza la palabra reservada *implements.*
    - Sintaxis:
        - **Implementación de una sola interfaz: (herencia simple de interfaces).**

```
    class Nombre de la Clase implements Nombre de la Interfaz

    {
```

Definición de propiedades de la clase;

Definición de métodos de la clase {Instrucciones};

Implementación de métodos de la interfaz {Instrucciones};

Implementación de métodos de la interfaz {vacío}; // Si no se va a usar un método de la interfaz debe dejarse vacío.

}

- **Implementación de varias interfaces: (herencia múltiple de interfaces)**

class Nombre de la Clase implements Nombre de la Interfaz1, Nombre de la Interfaz2, …, Nombre de la InterfazN

{

Definición de propiedades de la clase;

Definición de métodos de la clase {Instrucciones};

Implementación de métodos de la interfaz {Instrucciones}

}

- Ejemplo Interface:
  - **Creación de la interface.**

interface Volar {

public function Tipo();

public funtion Velocidad ($velocidad);

public function Aerodinamica();

}

- **Creación de una clase que implementa la interface.**

class Pajaro implements Volar{

public function Tipo(){

echo “Pájaro”;

}

public funtion Velocidad ($a){

```
        echo "Velocidad de vuelo: ". $vel;

    }

    public funtion Aerodinamica (){} // Método vacío que no se va a usar.

    }
```

- **Creación de objetos.**
  - $gorrion = new Pajaro();
  - echo $gorrion->Tipo;
  - echo $gorrion->Velocidad ("40 Km/h.");

- Ejemplo de polimorfismo con interfaces. (Otra clase con distinta implementación de los mismos métodos):
  - **Creación de una clase que implementa la interface.**

```
class Avion implements Volar{

    public function Tipo(){

        echo "Avión";

    }

    public funtion Velocidad ($b){

        echo "Velocidad de vuelo: ". $vel;

    }

}
```

  - **Creación de objetos.**
    - $irbusA300 = new Avion();
    - echo $airbusA300->Tipo;

echo $airbusA300->Velocidad ("900 Km/h.");

FORMULARIOS Y PHP.

- Los formularios son la forma que tiene el usuario de transmitirnos información, siendo, una de las principales funciones de PHP, recolectar los datos enviados desde un formulario HTML.
- Para gestionar formularios generalmente se requiere 2 páginas:
  - La que contiene el formulario.
  - La que procesa los datos cargados en el formulario.
- No obstante, también se puede utilizar una única página para incluir ambos contenidos.

Formas de utilizar PHP para gestionar formularios:

- Para procesar los datos de un formulario.
- Para crear el formulario, si éste debe incluir información generada dinámicamente.

Procesar un formulario usando PHP.

- Hay 3 métodos principales:
    - Incluir el formulario en una página web HTML e indicar el nombre del archivo PHP que procesará el formulario en el atributo "action" del formulario. De esta forma el formulario no contiene ningún elemento dinámico.
    - Incluir el formulario en un script PHP e indicar el nombre del archivo PHP que procesará el formulario en el atributo "action" del formulario. De esta forma el formulario puede contener elementos que se generen dinámicamente.
    - Incluir el formulario en un script PHP e indicar el nombre del mismo archivo PHP que procesará el formulario en el atributo "action". De esta forma, también el formulario puede contener elementos que se generen dinámicamente.

Atributos de formularios importantes para el procesamiento PHP.

- **action.**
    - Atributo de la etiqueta <form> que especifica a qué página se envían los datos del formulario para su procesamiento.
- **method**.
    - Atributo de la etiqueta <form> que establece de qué modo se enviaran los datos.
    - Valores:
        - **get**.
            - Con get los datos se envían junto con la URL de la página separados de ésta con una interrogación.
            - Con este método los datos son visibles por lo que no es muy apto para el envío de datos sensibles.
            - No permite tampoco el envío de muchos valores.
            - El método envía los datos usando la URL, es decir, se pasan variables (nombre+valor) concatenados con el operador &, la URL y el archivo de procesamiento unidos por una interrogación (?).
            - Ejemplo:
                - https://www.misitio.com/programas/procesamiento.php?nombre=Pepe&password=1234.....
        - **post.**
            - Envía los datos separados de la dirección de destino, por lo que no quedan a la vista
            - Permite el envío de datos muy grandes.
- **enctype**.

    - Atributo de la etiqueta <form> que define la codificación que se utilizará al enviar los datos.
- **name.**
    - Atributo utilizable en todos los campos o controles de un formulario.
    - Permite especificar un nombre para identificar los distintos controles.
    - Solo pueden consultarse datos de un formulario desde el lado del servidor que incorporen el atributo name.

Variables superglobales para formularios.

En realidad, son arrays superglobales al manejar formularios.

- **$_REQUEST.**
    - Mediante esta variable se usa para recopilar los datos enviados por un formulario HTML.
    - Básicamente es una combinación de las variables superglobales $_GET, $_POST y $_COOKIE.
    - Es en realidad un array asociativo en el que se almacenan automáticamente todos los datos del formulario.
    - La clave asociativa del array es el nombre del campo especificado en el atributo name para las etiquetas <input>, <textarea> y <select>.
    - El valor es el valor introducido en el campo.
    - Sintaxis:
        - $_REQUEST[“clave asociativa”];
    - Ejemplo:
        - <input type = “text” name = “contraseña”>
        - $_REQUEST[“contraseña”]; // contraseña es el valor del atributo name para el campo y se utiliza como clave asociativa en el array.
        - El resultado será el valor escrito en el cuadro de texto de campo creado.
- **$ _POST**.
    - Esta variable se usa para recopilar datos enviados mediante un formulario HTML con método post.
    - Es en realidad un array asociativo en el que se almacenan automáticamente todos los datos del formulario cuando este se envía con el método post.
    - La clave asociativa del array es el nombre del campo especificado en el atributo name para las etiquetas <input>, <textarea> y <select>.
    - El valor es el valor introducido en el campo.
    - Sintaxis:
        - $_POST[“clave asociativa”];
    - Ejemplo:
        - <input type = “text” name = “edad”>
        - $_REQUEST[“edad”]; // edad es el valor del atributo name para el campo y se utiliza como clave asociativa en el array.

        - El resultado será el valor escrito en el cuadro de texto de campo creado.
- **$ _GET.**
    - Esta variable se usa para recopilar datos enviados mediante un formulario HTML con método get.
    - Es en realidad un array asociativo en el que se almacenan automáticamente todos los datos del formulario cuando este se envía con el método get.
    - La clave asociativa del array es el nombre del campo especificado en el atributo name para las etiquetas <input>, <textarea> y <select>.
    - El valor es el valor introducido en el campo.
    - Sintaxis:
        - $_GET[“clave asociativa”];
    - Ejemplo:
        - <input type = “text” name = “nacionalidad”>
        - $_GET[“nacionalidad”]; // nacionalidad es el valor del atributo name para el campo y se utiliza como clave asociativa en el array.
        - El resultado será el valor escrito en el cuadro de texto de campo creado.
    - **URL.**
        - Todos los parámetros de una URL también se almacenan en el Array asociativo $_GET.
        - La clave asociativa es el nombre del parámetro.
        - El valor es el valor pasado junto con el parámetro.
        - Sintaxis:
            - https://www.sitio.dominio/ruta de acceso/archivo procesamiento.php?parametro1=valor 1&parametro2=valor 2&, …, &parámetro N = valor N
        - Ejemplo:
            - https://www.sitio.es/procesar.php?dni=99999999W&nombre=”Lola”&sueldo=1200.00
            - $_GET[“dni”] // Mostrará 99999999W.
            - $_GET[“nombre”] // Mostrará Lola.
            - $_GET[“sueldo”] // Mostrará 1200.00.
- **$ _FILES.**
    - Para poder subir un archivo al servidor es necesario añadir a la etiqueta <form> el atributo ***enctype*** con el valor ***multipart/form-data.***
    - Con esta variable/array asociativo se obtiene información sobre los archivos cargados y enviados con el formulario.
    - La información que se obtiene es relativa al nombre del archivo, tipo, tamaño, ruta en el servidor del archivo cargado y si se ha producido un error en su carga.
    - Valores contenidos en el array:
        - $_FILES[“valor name”]['name']

- Contiene el nombre original del archivo en el equipo local.

- $_FILES[“valor name”]['tmp_name']
  - Nombre temporal del archivo que almacena el fichero subido en el servidor.
  - Se muestra la ruta de acceso a la carpeta temporal el sitio.
  - Si es archivo temporal no se gestiona (renombra, mueve o copia), se elimina al terminar el script.
- $_FILES[“valor name”]['type']
  - Tipo MIME del archivo subido.
- $_FILES[“valor name”]['size']
  - Tamaño del archivo medido en bytes.
  - Tamaños de archivos modificables en PHP.ini dentro de la propiedad upload_max_filesize y también en MAX_FILE_SIZE cuyo valor no puede ser mayor que el de upload_max_filesize.
  - Máximo valor por defecto: 40 MB.
  - Valor 0 en upload_max_filesize permite subir archivos con cualquier tamaño.
- $_FILES[“valor name”]['error']
  - Tipo de error que puede ocurrido al subir el archivo.
  - Valores:
    - **0**:
      - Sin errores. Éxito en la subida.
    - **1** y **2**:
      - Tamaño superior al permitido para subir.
    - **3**:
      - Archivo cargado parcialmente.
    - **4**:
      - No se ha introducido ningún nombre de archivo.
    - **6:**
      - No hay directorio temporal.
    - **7**:
      - Error al escribir el archivo en el disco.
    - **8**:
      - Error en la extensión del archivo.

- **copy()**.
  - Función que permite copiar archivos.
  - La función devuelve ***true*** si la copia se ha realizado con éxito y ***false*** en caso contrario.
  - Si el archivo de destino ya existe, se sobrescribe.
  - Puede usarse una condición para preguntar si existen los archivos de origen o destino.
  - Sintaxis:

- copy (origen, destino);
- copy($_FILES[“valor name”]['tmp_name'],$_FILES[“valor name”]['name']); //_Si se usan las rutas especificadas en $_FILES cuando se sube un archivo al servidor.
- Origen:
  - Archivo a copiar.
- Destino:
  - Archivo nuevo (copia), en la ruta que se especifique.

- Ejemplo:

```
if (copy(“archivos/texto.pdf”, “copias/texto.pdf”))
{
        echo “Se ha copiado el archivo correctamente”;
}
else
{
        echo “Error al copiar el archivo”;
}
```

- Ejemplo confirmar si archivo de destino existe:

```
if (!file_exists(“archivos/datos.pdf”))
{
  //Se realiza la copia.
}
```

- Ejemplo confirmar si archivo temporal se ha enviado:

```
If($_FILES[“valor name”]['tmp_name']!=””)
{
  //Se realiza la copia.
}
```

- **move_uploaded_file** ()
  - Función que permite copiar archivos a su ubicación final.
  - Para realizar un movimiento hay que eliminar el archivo de su ubicación de origen.
  - Un archivo se borra con el método unlink().
  - La función devuelve ***true*** si el movimiento se ha realizado con éxito y ***false*** en caso contrario.
  - Si el archivo de destino ya existe, se sobrescribe.
  - Puede usarse una condición para preguntar si existen los archivos de origen o destino.
  - Sintaxis:
    - **move_uploaded_file** (origen, destino);
    - Origen:
      - Nombre del archivo subido.
    - Destino:
      - Destino del archivo a mover, incluyendo su nombre.
  - Ejemplo:

```
if (move_uploaded_file("archivos/texto.pdf",
"carpeta2/texto.pdf"))
 {
        echo "Se ha movido el archivo correctamente";
}
Else
{
        echo "Error al mover el archivo";
}
```

- Ejemplo confirmar si archivo de destino existe:

```
if (!file_exists("archivos/datos.pdf"))
{
  //Se realiza el movimiento.
}
```

unlink().

- Permite borrar un archivo.
- Sintaxis:
  - unlink ("nombre.extensión");
  - unlink ("ruta de acceso/nombre.extensión");
  - unlink (variable); // Variable que contenga la cadena que representa al archivo y su ubicación.
  - unlink($_FILES["valor atributo name"]["name"]);
- Ejemplos:
  - unlink ("foto.jpg");
  - unlink ("archivos/documentos/informe.pdf");
  - $archivo_borrar = "css/temas.css";
  - unlink($archivo_borrar); // Variable con la ruta y nombre del archivo.
  - unlink($_FILES["foto"]["name"]); // Borra el archivo cargado desde un formulario cuyo valor en el atributo *name* es foto.

Comprobación de campos vacíos.

- Si un campo se ha quedado vacío existirán en los arrays **$ _GET**, **$ _POST** o **$_REQUEST** claves asociativas sin ningún valor asociado.
- Se puede preguntar por campos vacíos usando una instrucción if...else.
- Sintaxis:

```
if ($_VARIABLESUPERGLOBAL["clave asociativa"] ==" ")
{
Instrucciones;
}
else
{
Otras instrucciones;
```

```
}
```

- Ejemplo:

```
if ($_REQUEST[“nombre”] ==” “)
{
 echo “<p>No has escrito un nombre</p>”;
}
else
{
echo “<p>Te llamas: “ . $_REQUEST[“nombre”]. </p>”;
}
```

Controles inexistentes.

- Puede ocurrir que se esté preguntando por campos inexistentes o que no han llegado con el formulario.
- Antes de realizar una acción se puede preguntar si existen con la función isset()
- Sintaxis:

```
if (isset($_VARIABLESUPERGLOBAL[“clave asociativa”]))
{
        Instrucciones;
}
else
{
        Otras instrucciones o no hacer nada;
}
```

- Ejemplo:

```
if (isset($_REQUEST[“nombre”]))
{
        echo “<p>Te llamas: “ . $_REQUEST[“nombre”]. </p>”;
}
```

Procesado por PHP de algunos controles de un formulario.

- **Botones de opción.**
  - Etiqueta <input>, atributo “type” con valor: “radio”.
  - Permiten seleccionar una opción entre varias excluyentes.
  - Para que sean excluyentes deben formare un grupo, que se crea incluyendo el mismo valor en el atributo “name” para todas las opciones.
  - Cuando se quiera acceder al valor del elemento a traes de variables superglobales, nos encontramos con el problema de que todas las opciones tienen el mismo valor en “name”.
  - Para averiguar que opción ha sido seleccionada se añade el atributo “value “con nombres distintos para cada opción y luego se pregunta por ellos con instrucciones condicionales.
  - Sintaxis:

    ```
    if ($_REQUEST [“valor del atributo name”] == “valor del atributo value”) { …. }
    ```

  - Ejemplo:

<input type “radio” name = ”opc” value =”uno”>Opcion 1
<input type “radio” name = ”opc” value =”dos”>Opcion 2
if($_REQUEST[“opc”] == “uno”) {…}
else if ($_REQUEST[“opc”] == “dos”) {…}

- **Casillas de verificación.**
  - Etiqueta <input>, atributo “type” con valor: “checkbox”.
  - Permiten seleccionar ninguna, una o varias opciones simultáneamente.
  - Problema:
  - Este campo se envía, sólo si se marca la casilla.
  - Para confirmar que una opción ha sido seleccionada y por tanto enviada, se puede utilizar el método isset().
  - Valores enviados:
    - Si está definido el atributo “value”, se envía su valor.
    - Si no está definido el atributo “value”, se envía la palabra “on”.
  - Sintaxis:

    if (isset($_REQUEST[“valor del atributo name”]) { …. }
  - Ejemplo:

    <input type “checkbox” name =”c1”>Opción 1
    <input type “checkbox” name =”c2” value “dos” ;>Opción 2
    if(isset($_REQUEST[“c1”]) {echo “Opción 1 elegida: “. ($_REQUEST[“c1”]}; // Devuelve “on”.
    if(isset($_REQUEST[“c2”]) {echo “Opción 1 elegida: “. ($_REQUEST[“c2”]}; // Devuelve “dos”.
- **Listas o menús desplegables.**
  - Etiquetas <select> y <option>.
  - Se pueden crear listas de selección simple o múltiple.
  - En las simples se envía un solo valor y en las múltiples, se crea un array con los valores enviados.
  - En las listas de elección múltiple se pueden cargar todos sus valores en otro array y usar éste en bucles for y foreach para mostrar las opciones elegidas.
  - Valores enviados:
    - Si está definido el atributo “value”, se envía su valor.
    - Si no está definido el atributo “value”, se envía el texto asociado a la casilla.
  - Sintaxis:
  - Selección simple: (Con y sin atributo “value”).

    $_REQUEST [“valor del atributo name”];
  - Selección múltiple: (Con y sin atributo “value”).

    $variable = $_REQUEST [“valor del atributo name”];
  - Ejemplo:

    echo “Opción 1 elegida: “ . $_REQUEST[“lista1”];
    $arraylista = $_REQUEST[“lista2”]; //No se ponen los corchetes del valor de name en la lista <select> con selección múltiple.

```
foreach ($arraylista as $datos)
{
        echo “Opción elegida: “ . $datos;
}
```

ARCHIVOS.

- En ocasiones puede ser necesario crear, guardar, leer o escribir archivos en la elaboración de sitios web.
- Para registrar o almacenar información en el servidor podemos utilizar archivos de texto y bases de datos.
- PHP incluye funciones para realizar las siguientes operaciones con archivos:
    - Confirmar existencia.
    - Creación y apertura.
    - Lectura.
    - Escritura.
    - Eliminación.
    - Cierre.
- **Secuencia de uso de funciones para manejar archivos.**
    - Abrir el archivo con *fopen()*.
    - Leer o escribir en el con *fread(), fgets() o fwrite().*
    - Al terminar de utilizar el archivo, cerrarlo con *fclose().*

CONSTANTES.

- **PHP_EOL**
    - Añade un salto de línea independientemente del sistema operativo.
    - Sintaxis:
        - PHP_EOL
    - Ejemplo:
        - $numeros = array(1,6,7,3,9,2,);
        - echo implode('', '.PHP_EOL, $numeros);
        - echo implode('',
”, $numeros);

FUNCIONES PARA MANEJO DE ARCHIVOS.

- **file_exists()**
    - Antes de manejar archivos puede ser necesario saber si éstos existen.
    - Comprueba o verifica la existencia de un archivo.
    - Devuelve un valor booleano (true, false).
    - Sintaxis:
        - file_exists(“ruta de acceso/nombre del archivo.extensión”);
    - Ejemplo:

```
if (file_exists(“datos.txt”))
{
        echo “El archivo existe”;
}
```

```
else
{
        echo “El archivo no existe”
};
```

- **fopen().**
  - Abre un fichero local o en el servidor actual o un archivo en servidores remotos usando una URL.
  - Si la cadena que representa el recurso a abrir comienza por http://, https://, ftp://, etc., se asume que es una URL.
  - Para poder abrir una URL, debe estar habilitada (on) la propiedad *allow_url_fopen* en el archivo *PHP.ini.*
  - Una URL puede abrirse para analizar y descomponer alguna cadena de datos que contenga, como texto**,** archivos de configuración**,** JSON**,** XML, etc., para que así pueda ser entendida y utilizada por un programa (parsing o parsear).
  - Si la acción se realiza correctamente, devuelve un puntero al archivo abierto, si no, devuelve false, por lo que suele utilizarse en estructuras condicionales.
  - Los archivos se abren para realizar operaciones de lectura o escritura en ellos.
  - Cuando se abre un fichero es necesario almacenarlo en un recurso para poder acceder a él y manejarlo, para ello, se crea una variable para guardar el valor devuelto por fopen().
  - Esta variable se usará después en otras funciones como fwrite(), fclose(), etc.
  - **Recurso**:
    - Tipo de dato especial que contiene una referencia hacia un recurso externo (stream, archivo o una conexión de base de datos).
    - Si que quiere mostrar el tipo de recurso, se puede usar la función get_resource_type($variable).
- <u>Sintaxis:</u>
  - fopen(“ruta de acceso/nombre archivo.extensión”, ”modo de apertura”);
  - **ruta de acceso**:
    - Puede ser local o remota, absoluta o relativa.
    - En Windows escapar la barra inclinada inversa e las rutas de acceso.
  - **modos de apertura**:
    - **r.**
      - Apertura para lectura.
      - Puntero al principio del archivo.
    - **r+.**

        - Igual que anterior, pero el archivo se abre para lectura y escritura.
        - Si el archivo no es escribible, abrirlo con r+ dará error.
    - **w.**
        - Se abre el archivo para escritura.
        - Puntero al principio del archivo
        - Elimina cualquier contenido del archivo.
        - Si no existe el archivo, se crea.
        - Es decir, escribe en un archivo nuevo o sobrescribe uno existente.
    - **w+.**
        - Igual que anterior, pero el archivo se abre para lectura y escritura.
    - **a.**
        - Abre un archivo para escritura.
        - Puntero al final del archivo.
        - Añade contenido al final del archivo.
        - Si el archivo no existe, se crea.
    - **a+.**
        - Igual que anterior, pero el archivo se abre para lectura y escritura.
    - **x.**
        - Crea un nuevo archivo para escritura sólo si no existe.
        - Si el archivo existe, se produce un error.
    - **x+.**
        - Igual que anterior, pero el archivo se abre para lectura y escritura.
    - **c.**
        - Abre un archivo para escritura.
        - Si no existe el archivo se crea.
        - Si existe, no se sobrescribe, ni da ningún error.
        - Puntero al principio del archivo.
    - **c+.**
        - Igual que anterior, pero el archivo se abre para lectura y escritura.
        - Cuando se manejan archivos binarios, como imágenes, hay que añadir una letra *b* después del modo de apertura como, por ejemplo: *rb* o *r+b*
- Ejemplos:
    - $fichero = fopen(“datos.txt”, ”r”);
    - $archivo = fopen(“c:\\ejercicios\\texto.txt”, ”w+”);

- **die().**
    - Muestra un mensaje y termina el script actual.

    - Permite gestionar errores o mostrar mensajes de error personalizados antes de detener la ejecución del script.
    - Equivalente a la función exit().
    - Sintaxis:
        - die(“Mensaje de error”);
    - Ejemplo:
        - fopen("prueba.txt", 'w') or die("Se produjo un error al crear el archivo");
- **fwrite()**
    - Permite escribir contenido en un archivo.
    - **Problemas de escritura:**
        - Disco lleno.
        - No se tienen permisos para crear o escribir en el archivo. (Modificar los atributos del fichero o de la carpeta donde éste esté almacenado, por si esta seleccionado *sólo lectura*.).
        - Otros.
    - Sintaxis:
        - fwrite(archivo, contenido, bytes);
        - **archivo.**
            - Archivo en el que se va a escribir. (Recurso creado con *fopen*).
        - **contenido.**
            - Cadena a escribir que puede ser un texto literal o una variable que lo contenga.
        - **bytes.**
            - Número máximo de bytes a escribir. (1 byte -> un carácter incluidos espacios en blanco y signos de puntuación).
            - Opcional.
    - Ejemplo:
        - $fichero_datos = fopen(“archivo.txt”,”w”);
        - $contenido = “Texto a incluir en el fichero”;
        - fwrite($fichero_datos, $contenido);
        - fwrite($fichero_datos, “Texto a incluir el fichero”);
- **fread()**
    - Lee un archivo abierto con *fopen* hasta el número especificado bytes o hasta el final.
    - Si no hay más datos que leer o se produce un error, devuelve *false*.
    - Sintaxis:
        - fread(archivo, bytes);
        - **archivo.**
            - Archivo que se va a leer. (Recurso creado con *fopen*).
        - **bytes.**
            - Número de bytes a leer.

- Si se quiere leer todo el archivo, hay que averiguar cuántos bytes tiene con la función filesize().

  - Ejemplo:
    - $fichero_datos = fopen("archivo.txt","r");
    - echo fread($fichero_datos,10); // Lee 10 bytes.
    - echo fread($fichero_datos, filesize("archivo.txt"); // Lee todo el fichero.

- **fgets()**
  - Lee y devuelve una línea de un archivo abierto con *fopen*.
  - Para leer más líneas se puede usar un bucle hasta llegar a la última línea y al final del fichero.
  - Si no hay más datos que leer o se produce un error, devuelve *false*.
  - Sintaxis:
    - fgets(archivo, bytes);
    - **archivo.**
      - Archivo que se va a leer. (Recurso creado con *fopen*).
    - **bytes.**
      - Número de bytes a leer.
      - Opcional.
  - Ejemplo:
    - $fichero_datos = fopen("archivo.txt","a+");
    - echo fgets($fichero_datos); // Muestra una línea del fichero.
    - while(!feof($fichero_datos))
      {
      echo fgets($fichero_datos); // Muestra todas las líneas hasta el final del fichero.
      };

- **feof().**
  - Comprueba si se ha alcanzado o no el final del fichero abierto con *fopen*. (Puntero al final del archivo).
  - Uso en condiciones y bucles para recorrer un fichero hasta el final.
  - Devuelve:
    - **true.**
      - El puntero ha alcanzado el final del fichero.
    - **false.**
      - El puntero no ha alcanzado el final del fichero.
    - **error.**
      - Se ha producido un error.
  - Sintaxis:
    - feof(archivo);
  - Ejemplo:
    - if(feof($fichero_datos))
    - {
      echo "fin del fichero";

```
};
```

- **filesize()**
  - Devuelve el tamaño de un archivo en bytes.
  - Como argumento hay que especificar el nombre del archivo, no un recurso abierto con fopen().
  - Sintaxis:
    - filesize(“archivo”);
  - Ejemplo:
    - echo filesize(“archivo.txt”);
- **fclose()**
  - Cierra un archivo tras su uso para liberar los recursos del sistema.
  - No es obligatorio, pero si aconsejable.
  - Sintaxis:
    - fclose(archivo);
  - Ejemplo:
    - fclose($fichero_datos);
- **file()**
  - Permite cargar el contenido de un archivo en un array.
  - Carga cada línea de texto en una posición distinta del array, incluyendo el carácter de fin de línea (EOL).
  - En caso de error devuelve false.
  - Sintaxis:
    - file(“ruta de acceso/nombre del archivo.extensión”);
  - Ejemplo:

```
$cadenas1 = file(“textos/datos.txt”);
$cadenas2 = file(“parrafos.txt”);
echo count($cadenas1); // Muestra cuantos elementos tiene el array (cuantas líneas se han cargado).
echo count($cadenas2); // Muestra cuantos elementos tiene el array (cuantas líneas se han cargado).
```

- **file_get_contents().**
  - Permite cargar el contenido textual de un archivo en una variable de tipo cadena.
  - En caso de error devuelve false.
  - Sintaxis:
    - file_get_contents(“ruta de acceso/nombre del archivo.extensión”);
  - Ejemplo:

```
$texto1 = file_get_contents(“textos/datos.txt”);
$texto2 = file_get_contents(“parrafos.txt”);
echo $texto1;
echo $texto1;
```

- **allow_url_fopen**.
  - Directiva que permite el acceso por URL a ficheros mediante funciones como file_get_contents() o fopen().

- Se configura en el archivo PHP.INI ajustando los siguientes valores:
  - **On**:
    - Activado.
  - **Off**:
    - Desactivado.
- Sintaxis:
  - allow_url_fopen = on
  - allow_url_fopen = off

- **allow_url_include.**
  - Directiva que permite el acceso por URL a ficheros mediante funciones como require(), require_once(), include() e include_once().
  - Se configura en el archivo PHP.INI ajustando los siguientes valores:
    - **On**:
      - Activado.
    - **Off**:
      - Desactivado.
    - Sintaxis:
      - allow_url_include = on
      - allow_url_include = off

CLASES Y FUNCIONES DE FECHA Y HORA.

- **getdate()**
  - Carga todos los parámetros de la fecha y hora actual en un array asociativo.
  - Claves**:**
    - **year.**
      - Años con 4 dígitos.
    - **month.**
      - Mes en texto
    - **mon.**
      - Mes en número.
    - **mday.**
      - Día en número.
    - **wday.**
      - Día de la semana en número.
      - Domingo = 0 hasta sábado = 6.
    - **yday**.
      - Día del año en número.
      - De 0 a 365.
    - **weekday**.
      - Día de la semana en texto.
    - **hours.**
    - **minutes.**
    - **seconds.**
    - **0.**

- Tiempo Unix.
- Cantidad de segundos que han transcurrido desde las 00:00:00 (UTC), del 1 de enero de 1970.
- Se usa para determinar la duración de ejecución de un determinado comando.

- Datos**:**
  - Los distintos valores de la fecha y hora actual.
- Sintaxis:
  - $array_asociativo = getdate();
- Ejemplo:

```
$actual = getdate();
print_r $actual; // Mostrar datos de la fecha con una función de salida.

foreach($actual as $periodo => $valor) // Mostrar datos con bucle foreach().
{
        echo $periodo. ":" .$valor;
}
```

- **date().**
  - Muestra la fecha y la hora actual con formato y como cadena de texto.
  - Formato:
    - Para aplicar el formato se utilizan distintos caracteres.
    - **Caracteres para formato de fechas**:
      - Números con ceros delante.
        - **d** (día), **m** (mes), **y** (años 2 dígitos), **Y** (años 4 dígitos).
      - Números sin ceros delante.
        - **j** (día), **n** (mes).
      - Nombre corto.
        - **D** (día), **M** (mes)
      - Nombre completo.
        - **l** (día), **F** (mes).
    - **Caracteres para formato de horas**:
      - Números con ceros delante.
        - **h** (horas 12), **H** (horas 24), **i** (minutos), **s** (segundos).
      - Números sin ceros delante.
        - **g** (horas 12), **G** (horas 24).
  - Sintaxis:
    - date (cadena con caracteres de formato, timestamp)
    - **timestamp**.
      - Marca temporal o de tiempo de Unix incluida como número entero.

- Es opcional. Si no se incluye, se usa time().

- Ejemplos:

```
echo date("d-m-Y"); // 20-07-2023.
echo date("d/m/Y"); // 20/07/2023.
echo date("l-F-Y"); // Sunday - july - 2023.
echo date("F"); // July;
echo date("Y"); // 2023.
echo date("H:i:s"); // 22:24:45.
echo date("H"); / 22.
```

- **checkdate().**
  - Comprueba la validez de una fecha.
  - Devuelve un valor booleano:
    - **true** si la fecha es válida.
    - **false** en caso contrario.
  - Sintaxis:
    - checkdate (mes, día, año)
    - parámetros**:**
      - **mes.**
        - En número de 1 a 12.
      - **día**
        - En número dependiendo del mes.
      - **año.**
      - En número en el rango 1 a 32767.
  - Ejemplos:
    - checkdate (5, 17, 2006); // true.
    - checkdate (2, 30, 2022); //false.
- **time()**
  - Devuelve la fecha actual en segundos en tiempo Unix.
  - Se puede incluir dentro de date() para obtener fechas diferentes a la actual.
  - Sintaxis:
    - time();
    - Para transformar el número de segundos a los de otra fecha distinta de la actual.
      - date(“Formato de fecha”, time() (+/- (Número de días * 24 horas * 60 minutos * 60 segundos)).
  - Ejemplos:
    - Hoy:
      - time(); // En segundos.
      - date(“d”/”m”/”Y”, time()); // Como fecha.
    - Semana anterior:
      - date(“d”/”m”/”Y”, time() - (7 * 24 * 60 * 60)).
    - Semana siguiente:

        - date(“d”/”m”/”Y”, time() + (7) * 24 * 60 * 60)).
- **date_default_timezone_set().**
    - Establece la zona horaria predeterminada que usan las funciones de fecha y hora.
    - Devuelve un valor booleano si la zona horaria es válida o no.
    - Sintaxis:
        - date_default_timezone_set(“zona horaria”);
        - **Zonas horarias**:
            - Consultar en https://www.php.net/manual/es/timezones.php.
    - Ejemplos:
        - $zona_horaria =”Pacific/Palau”;
        - date_default_timezone_set(“Asia/Seoul”);
        - date_default_timezone_set($zona_horaria”);
- **date_default_timezone_get().**
    - Devuelve la zona horaria predeterminada que usan las funciones de fecha y hora.
    - Zona horaria predeterminada inicial: Europe/Berlin.
    - Sintaxis:
        - date_default_timezone_get();
    - Ejemplos:
        - date_default_timezone_set(“Europer/Rome”);
        - $zona_horaria_predeterminada = date_default_timezone_get();
        - echo $zona_horaria_predeterminada; // Muestra = ”Europe/Rome”;
- **mktime().**
    - Devuelve el tiempo Unix en segundos (marca de tiempo timestamp), de una fecha determinada.
    - Sintaxis:
        - mktime(hora, minutos, segundos, mes, día, año);
    - Ejemplos:
        - mktime(0,0,0,8,12,1999);
- **Clase DateTime()**
    - Clase que permite representar fecha y horas.
    - **Constructores**:
        - **DateTime().**
            - Con el constructor vacío se crea un objeto con el momento temporal actual.
            - Sintaxis:
                - $nombre del objeto = new DateTime();
            - Ejemplo:
                - $ahora = new DateTime();
        - **DateTime**(“año, mes, día”**)**.

- Con una fecha en el constructor para inicializar el objeto, se puede cargar cualquier fecha.
- Sintaxis:
  - $nombre del objeto = new DateTime(“año, mes, día”**)**;
- Ejemplo:
  - $fecha = new DateTime(“2023-04-21”);

- **DateTime**(“Expresión”**)**.
  - Sintaxis:
    - $nombre del objeto = new DateTime(“Expresiones”**)**;
    - **Algunas expresiones**:
      - now.
        - Momento actual.
      - tomorrow.
        - Mañana.
      - noon.
        - Hoy, al mediodía.
      - last day of Mes Año.
        - Último día del mes y año especificado.
      - x days x hours ago.
        - Hace x días y x horas.
      - x days ago x hours.
        - Hace x días, x horas después.
      - +x weeks
        - Sumar x semanas al momento actual.
      - last day of this month.
        - Último día del mes
      - last nombre día of x months.
        - Último día de la semana de hace x meses.
  - Ejemplos:
    - $fecha1 = new DateTime(“now”);
    - $fecha2 = new DateTime(“3 days 12 hours ago”);
    - $fecha3 = new DateTime(“+7 weeks'”);

- **Métodos.**
  - **format().**
    - Permite dar formato a la fecha y/u hora contenidas en un objeto de tipo DateTime().
    - Devuelve un string.
    - Sintaxis:
      - objeto->format(“cadena con caracteres de formato”);
    - Ejemplo:

        - fecha1->format(“d-m-Y”); // 20-03-23.
- **diff().**
    - Calcula la diferencia entre 2 fechas.
    - Intervalos temporales:
        - Para especificar qué intervalo temporal debe mostrase (días, meses, horas, etc.), se usa la función format() con la siguiente sintaxis:
            - objeto->format(“%caracter de formato texto”);
            - Los caracteres de formato son los mismos que para la función date(), pero en este caso van precedidos por un sigo % para especificar el momento temporal a visualizar, que se muestra en número.
            - Se pueden incluir cadenas de texto para que la salida sea más inteligible.
            - Se puede incluir distintos momentos temporales.
    - Sintaxis:
        - $objeto_fecha1->diff($objeto_fecha2);
    - Ejemplos:
        - $diferencia = $objeto_fecha1->diff($objeto_fecha2);
        - echo $diferencia->format("%d días"); // 20 días.
        - echo $diferencia->format("%d días, %m meses"); // 20 días, 3 meses.
- **modify().**
    - Permite incrementar o decrementar una fecha u hora especificando el intervalo temporal a añadir o restar.
    - Las marcas de tiempo o intervalos se añaden como una cadena de texto.
    - Sintaxis:
        - $objeto_fecha->modify(“+/- número intervalo temporal”);
        - Intervalos:
            - day/days.
            - month/months.
            - year/years.
            - hour/hours.
            - minute/minutes.
            - second/seconds.
    - Ejemplos:
        - $fecha = new DateTime(“2022-06-18”);
        - $años = 5;

- $fecha->modify(“+ 5 months”);
- $fecha->modify(“- 12 days”);
- $fecha->modify(+ $años . “years”);
- $masmeses = $fecha->format(“Y,m,d”);
- $menosdias =$fecha-> format(“Y,m,d”);
- $masaños =$fecha->format(“Y,m,d”);
- echo $masmeses // Muestra 2022-11-12.
- echo $menosdias // Muestra 2022-06-06.
- echo $masaños // Muestra 2027-06-18.

- **date_modify().**
  - Alias del método modify().
  - Sintaxis:
    - date_modify(Objeto DateTime, “+/- número modificador intervalo temporal”);
  - Ejemplo:
    - $fecha1 = new DateTime();
    - date_modify($fecha1,“+ 5 days”);
    - echo date_format($fecha1, ”d-m-Y”).
- **date_format().**
  - Alias del método format().
  - Sintaxis:
    - date_format(Objeto DateTime, “cadena con caracteres de formato”);
  - Ejemplo:
    - $fecha2 = new DateTime();
    - date_modify($fecha2,“+ 2 months”);
    - echo date_format($fecha2, ”d-m-Y”).

CADENAS.

- Los string o cadenas, están compuestos por caracteres alfanuméricos que ocupan un byte cada uno.
- Tamaño máximo de un string en PHP: 2 gigabytes.
- Se encierran entre comillas simple o dobles.
- **Sintaxis heredoc.**
  - Sintaxis que funciona igual que las comillas dobles, pero especificando la cadena de forma diferente.
  - Sintaxis:

```
<<<Identificador
Texto de la cadena línea 1
Texto de la cadena línea 2
Texto de la cadena línea N
Identificador;
```

- **Operador <<<.**

- Indica que se va a usar la sintaxis heredoc.

▪ **Identificador**.

- Expresión o abreviatura para especificar inicio y final de la cadena. (Se puede indicar la que se quiera).
- El identificador solo puede contener caracteres alfanuméricos y guiones bajos.
- El primer identificador da comienzo a la cadena y se escribe a continuación de <<<.
- El último, en la línea final seguido de un punto y coma (;) y debe coincidir en nombre con el primero.

○ Ejemplo:

```
$texto = <<<TEXTO
Texto escrito
en varias
líneas.
TEXTO;
```

FUNCIONES DE STRING.

- Conjunto de funciones que permiten manejar y manipular cadenas de caracteres.
- Algunas funciones:
  - **htmlentities().**
    - Muestra los caracteres especiales que utiliza HTML en lugar de su interpretación, es decir, convierte todos caracteres susceptibles de serlo en entidades HTML.
    - Sintaxis:
      - htmlentities("cadena de texto");
    - Ejemplo:
      - $texto = "Aire acondicionado. ";
      - echo "$texto"; // Muestra Aire **acondicionado.**
      - $texto = htmlentities($texto);
      - echo "$texto"; // Aire acondicionado. "**.**
  - **trim().**
    - Elimina espacios vacíos, saltos de tabulación, saltos de línea, etc., delante y detrás de una cadena.
    - Las funciones *ltrim()* y *rtrim()* son similares. La primera elimina los espacios en blanco delante de la cadena y la segunda, los de detrás.
    - Sintaxis:
      - trim("cadena de texto");
    - Ejemplo:
      - $texto = " Texto ";
      - var_dump("$texto"); // Muestra los espacios en blanco.
      - $texto = trim($texto);
      - var_dump("$texto"); // Elimina espacios sobrantes.

- **nl2br().**
    - Muestra los saltos de línea incluidos en un texto.
    - Sintaxis:
        - nl2br(“cadena de texto”);
    - Ejemplo:
        - $texto = “ Texto ”;
        - var_dump(“$texto”); // Muestra los espacios en blanco.
        - $texto = trim($texto);
        - var_dump(“$texto”); // Elimina espacios sobrantes.
- **strtolower() y mb_strtolower().**
    - Convierte un texto a minúsculas.
    - Algunos caracteres (tildes, eñes, etc.), pueden no convertirse correctamente por utilizarse varios bytes para representarlos.
    - Para la conversión correcta, se usan funciones multitbyte (empiezan por mb).
    - Sintaxis:
        - strtolower(“cadena de texto”);
        - mb_strtolower(“cadena de texto”);
    - Ejemplos:
        - $texto = “CAMIÓN”;
        - print(strtolower(“$texto”)); // Muestra: camiÓn. (No convierte bien un carácter acentuado).
        - print(mb_strtolower(“$texto”)); // Muestra: camión. (Conversión correcta del carácter acentuado).
- **strtoupper() y mb_strtoupper().**
    - Convierte un texto a mayúsculas.
    - Algunos caracteres (tildes, eñes, etc.), pueden no convertirse correctamente por utilizarse varios bytes para representarlos.
    - Para la conversión correcta, se usan funciones multitbyte (empiezan por mb).
    - Sintaxis:
        - strtoupper(“cadena de texto”);
        - mb_strtoupper(“cadena de texto”);
    - Ejemplos:
        - $texto = “camión”;
        - print(strtoupper(“$texto”)); // Muestra: CAMIóN. (No convierte bien un carácter acentuado).
        - print(mb_strtoupper(“$texto”)); // Muestra: CAMIÓN. (Conversión correcta del carácter acentuado).
- **ucfirst().**
    - Convierte a mayúsculas el primer carácter de una frase.
    - Sintaxis:
        - ucfirst(“cadena de texto”);
    - Ejemplo:

        - $texto = “el planeta Saturno”;
        - print(ucfirst(“$texto”)); // Muestra: El planeta Saturno.
- **ucwords().**
    - Convierte a mayúsculas el primer carácter de cada una de las palabras de una frase.
    - Sintaxis:
        - ucwords(“cadena de texto”);
    - Ejemplo:
        - $texto = “el planeta Saturno tiene anillos”;
        - print(ucwords(“$texto”)); // Muestra: El Planeta Saturno Tiene Anillos.
- **number_format().**
    - Aplica formato numérico en forma de cadena.
    - Sintaxis:
        - number_format (número con decimales a redondear).
            - Número sin decimales con formato numérico anglosajón.
        - number_format (número con decimales a redondear, número de decimales tras redondeo).
            - Número en formato anglosajón con el número de decimales especificado.
        - number_format (número con decimales a redondear, número de decimales tras redondeo, separador de decimales, separador de millares).
            - Número en formato personalizado con el número de decimales especificado.
    - Ejemplo:
        - $cantidad = 3,023.86765;
        - $texto = numer_format ($cantidad, 2, “.”,”,”);
        - echo $texto; // Muestra: 3.023,87.
- **implode().**
    - Devuelve todos los elementos del array especificado unidos en una cadena usando un determinado separador.
    - Si no se incluye un separador, se muestran todos los datos juntos en la misma línea sin espacio que los separe.
    - Para comprobar su funcionamiento, hay que incluir esta función dentro de alguna de las funciones de salida de datos como echo (), print(), print_r(), var_dump(), etc.
    - Sintaxis:
        - implode(“separador”, nombre del array);
    - Ejemplo:
        - print (implode (“, “, $numeros)); // Muestra el contenido del array $numeros en una línea separando sus datos con una coma y un espacio en blanco.

        - print (implode (“
“, $numeros)); // Muestra cada dato del array $numeros en una línea distinta.

- **explode().**
    - Crea un array con las subcadenas obtenidas al dividir una cadena mayor usando un determinado delimitador.
    - Sintaxis:
        - explode(“delimitador”, cadena de texto”);
    - Ejemplo:
        - $texto = “El señor de los anillos”;
        - $subcadenas = explode(“ ”, $texto”); // El delimitador es un espacio en blanco. Cadena separada en palabras.
        - Echo $ subcadenas[1]; // Muestra: señor.
- **strlen() y mb_strlen().**
    - Muestra la cantidad de caracteres que tiene una cadena, incluyendo espacios en blanco y signos de puntuación.
    - Como algunos caracteres (tildes, eñes, etc.), utilizan varios bytes para ser representados, el número final de caracteres puede ser mayor que el total real, por ello, es necesario usar una función multitbyte para contabilizarlos. (empiezan por mb).
    - Sintaxis:
        - strlen(“cadena de texto”);
        - mb_strlen(“cadena de texto”);
    - Ejemplos:
        - $texto = “Astronomía para aficionados”;
        - echo “Cantidad de caracteres: “. strlen($texto”); // Muestra 28, por la tilde.
        - echo “Cantidad de caracteres: “. mb_strlen($texto”); // Muestra 27.
- **str_word_count().**
    - Muestra la cantidad de palabras que tiene una cadena.
    - Sintaxis:
        - str_word_count(“cadena de texto”);
    - Ejemplos:
        - $texto = “Astronomía para aficionados”;
        - echo “Cantidad de palabras: “. str_word_count($texto”); // Muestra 3.
- **str_replace().**
    - Permite reemplazar un texto por otro.
    - Permite reemplazar unos caracteres por otros, estando los caracteres a reemplazar y los de reemplazo incluidos en un array.
    - Sintaxis:
        - str_replace(“subcadena a reemplazar, “subcadena de reemplazo”, “cadena de texto”);

        - str_replace(“array con los caracteres a reemplazar, “array con los caracteres de reemplazo”, “cadena de texto”);
    - Ejemplos:
        - $texto = “Buenos días”;
        - $letras1 = [“o”, ”í”];
        - $letras2 = [“a”, ”i”];
        - echo “Saludo: “. str_replace(“días”, ”noches”, $texto); // Muestra: Buenas noches.
        - echo “Saludo: “. str_replace($letras1, $letras2, $texto); // Muestra: Buenas dias.
- **str_contains().**
    - Busca una subcadena dentro de una cadena.
    - Devuelve un valor booleano si la subcadena está incluida, o no, dentro de la cadena.
    - Se puede usar dentro de una condición.
    - Distingue entre mayúsculas y minúsculas.
    - Sintaxis:
        - str_contains(“cadena de texto”, ”subcadena buscada”);
    - Ejemplos:
        - $texto = “Nave espacial”;
        - echo str_contains($texto, “espacial”); // Devuelve 1 (true).
        - echo str_contains($texto, “Espacial”); // Devuelve false.
        - Con condición:

```
If(str_contains($texto, “espacial”)
{
        echo “La palabra espacial está incluida en la frase”;
}
else
{
        echo “La palabra espacial no está incluida en la frase”;
}
```

- **strrev().**
    - Muestra el texto en orden inverso.
    - Sintaxis:
        - strrev(“cadena de texto”);
    - Ejemplos:
        - $texto = “Telefono”;
        - echo “Texto al revés: “. strrev($texto”); // Muestra *onofeleT*.
- **strpos().**
    - Muestra la posición en la que se inicia una subcadena dentro de una cadena.
    - Devuelve *false* si no se encuentra la subcadena.

- Sintaxis:
  - strpos(“cadena de texto”, “subcadena”, ”offset”);
  - **offset**.
    - Opcional.
    - Número entero que indica la posición a partir de la cual, se comienza la búsqueda.
    - Si el valor es positivo se comienza desde el principio de la cadena, teniendo en cuenta que el primer carácter ocupa la posición 0.
    - Si el valor es negativo, se comienza desde el final.
    - Puede utilizar con un bucle para encontrar todas las coincidencias de la cadena buscada y obtener la posición en la cada una de ellas comienza.
- Ejemplo:
  - $texto = “Anillos de Saturno”;
  - $posicion = strpos($texto, “Saturno”);
  - echo “Posición de inicio de la palabra Saturno: “. $posicion; // Muestra: 11.

- **substr()**
  - Devuelve o extrae una subcadena de una cadena mayor.
  - Sintaxis:
    - substr(“cadena de texto”, posición de inicio, caracteres);
    - **cadena**.
      - Texto del que se va a extraer una subcadena.
    - **posición de inicio**.
      - Carácter no incluido a partir del cual se comienza la extracción.
      - Si el valor es positivo se comienza desde el principio de la cadena, teniendo en cuenta que el primer carácter ocupa la posición 0.
      - Si el valor es negativo, se comienza desde el final.
      - Si el valor es mayor que la longitud de la cadena la función devuelve false.
    - **caracteres**.
      - Número de caracteres a partir de la posición inicial que se extraerán.
  - Ejemplos:
    - $cadena = “En un lugar de la Mancha”;
    - $subcadena = substr($cadena,6,2);
    - echo $subcadena; // Muestra lu.
- **substr_replace()**
  - Reemplaza una subcadena con otra.
  - Sintaxis:

        - substr_replace(“cadena de texto”, “subcadena de reemplazo”, posición, caracteres);
        - **posición**.
            - Carácter no incluido a partir del cual se comienza la sustitución.
            - Si el valor es positivo se comienza desde el principio de la cadena, teniendo en cuenta que el primer carácter ocupa la posición 0.
            - Si el valor es negativo, se comienza desde el final.
            - Si el valor es mayor que la longitud de la cadena, la subcadena se inserta al final de la cadena.
        - **caracteres**.
            - Número de caracteres a partir de la posición inicial que se reemplazarán.
            - Si no se incluye este valor, se sustituyen todos los caracteres hasta el final de la cadena.
            - Un valor negativo, representa el número de caracteres que se reemplazaran desde el final.
    - Ejemplos:
        - $cadena = “En un lugar de la Mancha”;
        - $subcadena = substr_replace($cadena, “Sierra”, 18);
        - echo $subcadena; // Muestra: En un lugar de la Sierra.
        - $subcadena = substr_replace($cadena, “ta”, 21, 3);
        - echo $subcadena; // Muestra: En un lugar de la Manta.
- **substr_count().**
    - Muestra la cantidad de ocurrencias de una subcadena en una cadena.
    - Distingue entre mayúsculas y minúsculas.
    - Sintaxis:
        - substr_count(“subcadena de texto”, cadena de texto”);
    - Ejemplos:
        - $texto = “Hace solo los días de sol”;
        - echo “Número apariciones: “. substr_count(“sol”, $texto); // Muestra 2.
- **strcasecmp().**
    - Compara dos cadenas entre sí en modo binario para confirmar, o no, que son iguales.
    - No distingue entre mayúsculas y minúsculas.
    - La función *strcmp()* es equivalente a strcasecmp(), sólo que si es sensible a mayúsculas y minúsculas.
    - Si diferencia palabras con o sin tilde.
    - Resultados de la comparación:
        - **0**:

        - Ambas cadenas son iguales.
      - **Número negativo**:
        - Las cadenas no son iguales.
        - La primera cadena es menor que la segunda.
        - Número que indica la separación alfabética de los caracteres de ambas cadenas.
      - **Número positivo**:
        - Las cadenas no son iguales.
        - La primera cadena es mayor que la segunda.
        - Número que indica la separación alfabética de los caracteres de ambas cadenas.
    - Sintaxis:
      - strcasecmp("cadena de texto 1", "cadena de texto 2");
    - Ejemplo:
      - $texto1 = "Marte";
      - $texto2 = "marte";
      - $texto3 = Jupiter";
      - echo "Comparación 1: ". strcasecmp($texto1, $texto2); // Muestra: 0, ambas cadenas son iguales.
      - echo "Comparación 2: ". strcasecmp($texto1, $texto3; // Muestra: 3, ambas cadenas son distintas y además, la primera es mayor que la segunda.
      - echo "Comparación 3: ". strcmp($texto1, $texto2); // Muestra: -1, ambas cadenas son distintas por la distinción mayúsculas-minúsculas que hace la función strcmp().

EXCEPCIONES.

- Situaciones que ocurren mientras se ejecuta un programa que detienen o alteran el flujo normal de éste.
- Si no se detecta una excepción puede causar un error fatal.
- Todas las excepciones son instancias u objeto de esta clase Exception.
- La clase Exception se puede extender, por lo que se pueden crear subclases a partir de ella para personalizar excepciones.
- Hay que usarlas para casos excepcionales, no parar errores comunes que ocurren frecuentemente y que se pueden gestionar de otras formas, como con mensajes de error genéricos.
- **Lanzar y Capturar una excepción.**
  - **Bloque try.**
    - Incluye el código que puede lanzar una excepción.
    - Si se lanza una excepción en este bloque, se interrumpe el flujo normal del programa y se pasa al bloque catch.
  - **Bloque catch.**
    - Incluye el código para capturar una excepción.
    - Este bloque se encarga de gestionarla.

        - Se pueden incluir varios bloques catch, cada uno encargado de capturar una excepción de un tipo distinto.
    - **Bloque finally**.
        - Este bloque se puede utilizar o no.
        - Si se usa, puede incluirse después o en lugar vez de los bloques catch.
        - Su código siempre se ejecutará independientemente de que haya o no ocurrido una excepción.
    - **throw.**
        - permite lanzar una excepción a capturar con catch.
        - Se crea una instancia de la clase Exception, por lo que a través de ella se pueden usar las propiedades y métodos de Exception.
        - El nombre por defecto para la instancia es $e.
        - Sintaxis:
            - throw new Exception ("Mensaje de Error Personalizado");

- Sintaxis de try … catch… finally:

```
try {
    // Código que puede producir una excepción;
} catch (Exception $e) {
    // código para gestionar o manejar la excepción;
}
[finally{
    // Código que se ejecutará se produzca o no la excepción;
}]
```

- Ejemplo:

```
function sumar2($num1, $num2){
  if(is_numeric($num1) and is_numeric($num2))
  {
      return $num1 + $num2;
  }
  else
  {
      throw new Exception("Los valores a sumar tienen que ser numéricos.");
      return 0;
  }
 }
 try{
      echo "Resultado de la suma: ".sumar2("hola", 4);
 }catch(Exception $e)
{
      echo $e->getMessage();
 }
```

- **Métodos.**
    - **getLine().**
        - Muestra el número de línea en el que se creó la excepción.
        - Sintaxis:

        - objeto excepción->getLine().
    - Ejemplo:
        - $e->getLine();
- **getMessage().**
    - Devuelve el mensaje de la excepción.
    - Sintaxis:
        - objeto excepción->getMessage().
    - Ejemplo:
        - $e->getMessage();
- **getFile().**
    - Muestra el fichero y su ruta de acceso, en el que se creó la excepción.
    - Sintaxis:
        - objeto excepción->getFile().
    - Ejemplo:
      $e->getFile();

COOKIES.

- **Concepto**
    - Archivo de pequeño tamaño con información enviada por un sitio web que se guarda en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del navegador.
    - Mecanismo por el que se almacenan datos en el navegador remoto para monitorizar o identificar a los usuarios que vuelvan a un sitio web (manual online PHP).
- **Funciones.**
    - Recordar accesos o visitas previas.
    - Conocer hábitos navegación de usuario para fines publicitarios o de otro tipo, lo que genera desconfianza en torno a la privacidad.
    - En general, identificar a un usuario que visita un sitio web y llevar un registro de su actividad en el mismo.
- **Tipos.**
    - **Cookies de sesión.**
        - Son las más comunes y también las más efímeras, ya que se eliminan cuando el usuario cierra el navegador.
    - **Cookies persistentes.**
        - No se borran al cerrar el navegador, sino al llegar su fecha de caducidad o cuando el usuario elimina sus datos de navegación.
        - Se usan para guardar información sobre el comportamiento de un usuario en un sitio web durante un periodo determinado.
    - **Cookies seguras.**
        - Se usan sólo en conexiones HTTPS y guarda los datos encriptados para minimizar vulnerabilidad ante ataques.
    - **Cookies zombis.**

        - Se almacenan en el sistema y no en el navegador, por lo que se puede acceder a ellas independientemente del navegador usado.
        - Son capaces de copiarse a sí mismas después de ser borradas, y a veces se usan con fines ilegales.
- **Características.**
    - Solo pueden ser leídas por los sitios que las han creado.
    - Su tamaño máximo es de 4 KB.
    - El límite de cookies por dominio depende del navegador. Suele estar en 20 cookies.
    - El límite de cookies almacenadas en el ordenador cliente suele ser unas 300. Al sobrepasar ese número, se eliminarán cookies antiguas para almacenar las nuevas.

CREACIÓN DE UNA COOKIE.

- **setcookie().**
    - Función que permite crea una cookie.
    - Sintaxis:
        - setcookie("nombre", "valor", "duración", "ruta", "dominio", "seguridad", "solo http").
        - Parámetros:
            - **nombre.**
                - Nombre dado a la cookie.
            - **valor**.
                - Valor de la cookie que se almacena en el ordenador del cliente.
            - **duración.**
                - Tiempo en que expira la cookie.
                - Es opcional, si no se incluye, la cookie expira al cerrar la sesión.
                - Otros valores:
                    - **0**.
                        - La cookie expira al cerrar la sesión.
                        - Si no se incluye su duración, también expira al cerrar la sesión.
                    - **Tiempo Unix.**
                        - Se usa una marca de tiempo Unix en segundos de modo que, para establecer un periodo se puede emplear la función time():
                            - time()+60 //cookie expira en 1 minuto.
                            - time()+(60*60*24*30) //cookie expira en 30 días.
                            - time()+(60*60*24*365) //cookie expira en un año.

- **ruta.**
  - Indica qué directorio tendrá acceso a la cookie.
  - Es opcional, si no se incluye, la cookie sea almacena en el directorio actual donde está el archivo en el que la cookie se ha configurado.
  - Todos los scripts contenidos en el directorio especificado y en sus subdirectorios, podrán leer la cookie, por ejemplo, si una cookie se almacena en un directorio denominado "/datos/", estará disponible en esa carpeta y todas sus subcarpetas, es decir, sólo se enviará al servidor si el navegador solicita un archivo de la carpeta datos o de sus subcarpetas.
  - Si se incluye "/" se almacena en la raíz y está disponible en todo el dominio. Es decir, la cookie se crea a nivel del sitio web y se enviará al servidor independientemente de la localización del archivo solicitado en éste.
- **dominio**.
  - Opcional, si no se especifica solo está disponible para el domino desde donde se crearon.
  - Si se especifica, se indica para qué dominio o subdominio estará disponible y se enviará la cookie.
- **seguridad**.
  - **true.**
    - Del lado cliente, sólo se enviará si se ha establecido una conexión **https.**
    - Del lado servidor, solo se creará si existe una conexión segura**.**
  - **false.**
    - Se enviará o creará, según el caso, sin una conexión https.
- **sólo http**.
  - **true.**
    - La cookie solo está disponible vía http, es decir, lenguajes del lado cliente no pueden acceder a ellas.
    - Valor por defecto.
  - **false.**
    - Está disponible para lenguajes del lado cliente como JavaScript.

- Ejemplos:
  - $nombre="usuario";
  - $valor=pepe;
  - $caducidad=time()+(60*60); // Una hora de duración;

- setcookie(“nombre”, ”juan”, time()+(3600 * 24 * 365)) // Duración un año.
- setcookie($nombre, $valor, $caducidad); Duración una hora.
- setcookie($nombre, $valor, $caducidad,”/cookies/”,”cosas.com”, false, true); // La cookie sólo se enviará al servidor si el navegador solicita un archivo de cosas.com/cookies/, además, se creará sin conexión segura y estará sólo disponible del lado del servidor.

LECTURA DE UNA COOKIE.

- Para leer las cookies la variable superglobal o array **$_COOKIE**, que permite para extraer los valores de la cookie**.**
- Sintaxis:
  - $variable = $COOKIE[“nombre”];
  - Se usa el nombre de la cookie para obtener su valor.
- Ejemplo:
  - $usuario = $COOKIE[$nombre];
- **Comprobar si existe una cookie.**
  - Uso de la función isset() preguntando con una condición.
  - Ejemplo**:**

```
if( isset( $_COOKIE[“nombre”]) )
{

    $usuario = $COOKIE[“nombre”]; // Carga en una variable de su valor si la cookie existe.
}
else
{
        setcookie(($usuario, ”juan”, time()+(3600 * 24); // Creación de la cookie si no existe.
}
```

ELIMINACIÓN DE UNA COOKIE.

- Para eliminar una cookie, se crea una nueva cookie con el mismo nombre y se establece su tiempo de expiración a un momento temporal pasado o se le asigna un número negativo.
- Una cookie también desaparece si no se incluye el argumento de tiempo o éste es igual a cero.
- Ejemplos:
  - setcookie($nombre, $valor, time() - 1); // Eliminación de la cookie por valor negativo.
  - setcookie($nombre, $valor, time() - (60*60); // Eliminación de la cookie por tiempo negativo.
  - setcookie($nombre, $valor); // Eliminación de la cookie al cerrar sesión.
  - $caducidad = 0;

- setcookie($nombre, $valor, $caducidad); // Eliminación de la cookie al cerrar sesión.

SESIONES.

- Una página web es un documento independiente, por lo que en principio dos scripts PHP no pueden compartir información.
- Formas de enviar información de una página a otras son:
  - Uso de formularios.
  - Envió de información en cabeceras.
  - Uso de variables comunes (variables de sesión) en programas distintos.
- **Características**:
  - Permiten que diferentes páginas accedan a una variable común, el array $_SESSION.
  - La información sobre las sesiones se mantiene en el servidor hasta que se cierra la sesión por parte del usuario o porque finalizado un determinado tiempo.

S_SESSION.

- Array que almacena información a través de las peticiones (*requests),* que un usuario hace durante su visita a un sitio web o aplicación.
- La información guardada en una sesión puede llamarse en cualquier momento mientras la sesión esté abierta.
- **Uso de sesiones**.
  - **Creación o apertura de sesión.**
    - **session_start()**
      - Inicia una nueva sesión o reanuda la existente.
      - Si la sesión no existe, se crea y se le asocia un Id único.
      - Si ya existe, la página se una a la sesión existente y así tiene acceso a la información vinculada a la sesión. Es decir, todas las páginas que quieran guardar datos en $_SESSION o leer datos de $_SESSION deben comenzar con la función session_start().
      - Esta función debe usarse antes de enviar el contenido de la página, si no, no se creará la sesión.
      - Se ejecuta una sola vez al principio de todo el código.
  - **Utilización de la sesión.**
    - Cuando la sesión está creada, las páginas solicitadas por el mismo navegador pueden guardar y recuperar información en el servidor.
    - Esta información se asocia a un identificador de usuario único, por lo que otros usuarios no pueden acceder a ella.
    - La información se conserva hasta que se destruye la sesión.
    - **session_id().**

- Muestra el id de la sesión actual, o una cadena vacía (“ ”), si éste no existe.
- Sintaxis:
  - session_id();
- Ejemplo:
  - echo session_id();

- **Eliminación o cierre de sesión.**
  - La información se conserva hasta que se destruye la sesión.
  - **session_detroy().**
    - Elimina toda la información asociada la sesión actual excepto:
      - Las variables de sesión, que se pueden volver a utilizar llamando de nuevo al método session_start().
      - La cookie de sesión.
    - Sintaxis:
      - session_destroy();
    - Ejemplo:
      - session_destroy();
  - **session_unset().**
    - Elimina sólo las variables de sesión, pero no la sesión, de modo que se truncan los datos.
    - Sintaxis:
      - session_unset();
    - Ejemplo:
      - session_unset();
    - También se pueden eliminar las variables de sesión cargando un array vacío en $_SESSION:
      - $_SESSION = array();