

PHP

Elementos del lenguaje

Código PHP

- El servidor procesa código PHP en ficheros con extensión **.php**
 - Un fichero .php puede contener texto de varios tipos:
 - Código PHP
 - Código HTML
 - Código JavaScript
- El código PHP normalmente está embebido en texto HTML
 - **<?php ... ?>**
 - Sintaxis recomendada
 - **<? ... ?>**
 - Short tags: Requiere que esté habilitada con la propiedad *short_open_tag on* en el fichero de configuración php.ini
 - **<script language="php"> ... </script>**
 - Como scripts en HTML (poco habitual)
 - **<% ... %>**
 - Estilo de ASP (no válido a partir de PHP 5.3)
 - Requiere que esté habilitada con la propiedad *asp_tags on* en el fichero de configuración php.ini

Código PHP

- Evaluación de expresiones en línea
 - `<?= expresión ?>`
 - Equivale a `<?php echo(expresión) ?>`

```
<html>
<head>
    <title>Evaluación de expresiones en línea</title>
</head>
<body>
    <!-- Expresión en línea -->
    <p>2+2= <?= 2 + 2 ?></p>

    <!-- Forma habitual de incluir código PHP -->
    <p>2+2= <?php echo(2 + 2); ?></p>
</body>
</html>
```

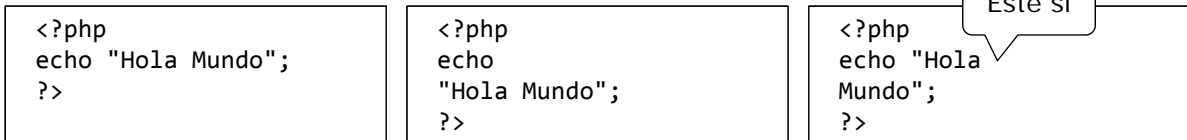
Inclusión de ficheros PHP

- Se pueden incluir otros ficheros con funciones de inclusión PHP:
 - `require('/directorio/fichero');`
 - `include('/directorio/fichero');`
 - `require_once('/directorio/fichero');`
 - `include_once('/directorio/fichero');`
 - `require` produce un error fatal y la terminación del script si falla
 - `include` solo produce un warning
 - `include_once` y `require_once` para incluir definiciones estáticas
- El directorio desde donde se buscan los includes se define en la directiva **`include_path`** del fichero *php.ini*

```
<html>
<?php
require ($_SERVER['DOCUMENT_ROOT'].'cabecera.php');
?>
<body>
<p>La cabecera de este documento la ha generado un programa PHP.
</p>
</body>
</html>
```

Sintaxis de PHP

- PHP distingue entre mayúsculas y minúsculas
- **Instrucciones**
 - Todas las instrucciones acaban con ;
 - Solo la última instrucción puede no acabar en ;
 - Se recomienda escribir cada instrucción en una línea
 - Aunque pueden ir en varias líneas
 - Los espacios y las líneas en blanco no se consideran

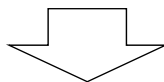


- **Comentarios**
 - // o # para comentarios hasta el final de la línea
 - /* ... */ para comentarios en varias líneas

Generación de código HTML

- Indistintamente con **echo** o **print**
void **echo**(string argument1 [, ...string argumentN])
int **print**(argument)
 - Son similares salvo que print
 - Solo tiene un argumento (echo puede tener varios)
 - print devuelve 1 (significa que ha generado la salida)

```
$usuario = "Juan";  
echo "<p>Bienvenido $usuario</p>";  
print "<p>Bienvenido $usuario</p>";  
print("<p>Bienvenido $usuario</p>");
```



```
<p>Bienvenido Juan</p><p>Bienvenido Juan</p><p>Bienvenido Juan</p>
```

Generación de código HTML

- Cuando hay varios argumentos a los que se les quiere aplicar un formato: **printf**

integer **printf**(string format [, mixed args])

- Tipos:

- %b número binario
- %c carácter ASCII
- %d número entero
- %f número en coma flotante
- %o número en octal
- %s string
- %u número decimal sin signo
- %x hexadecimal

```
printf("%d kilos de caramelos cuestan %.2f euros", 3, 27.90);
```

3 kilos de caramelos cuestan 27.90 euros

- **sprintf()** hace lo mismo pero genera un string que se puede asignar a una variable

Variables

- Todas las variables empiezan con el signo '\$'
\$identificador
 - El identificador comienza por letra o subrayado '_'
 - El resto de caracteres pueden ser letras, números o subrayado '_'
 - El identificador tiene en cuenta mayúsculas y minúsculas
- Se definen automáticamente en su primera utilización
- No se declara el tipo de las variables
 - Se puede averiguar con **gettype()** o **var_dump()**
- Las variables se pueden asignar
 - Por valor
 - Por referencia (con **&**)
 - Un cambio en la referencia también se aplica a la variable original

```
$x = 'equis';  
$_x = &$x;      // referencia a $x  
$_x = 'x';  
echo $x;        // x  
echo $_x;       // x
```

Variables

■ Ámbito de variables

■ Local

- Una variable definida en una función está limitada a dicha función
- Se elimina al acabar la ejecución de la función
- Salvo si la variable se declara como **static**
 - Las variables static solo se usan en la función
 - En las siguientes invocaciones a la función preservan su valor anterior

■ Global

- Una variable definida fuera de una función
- Se pueden definir en una parte y usarse en otra
 - Pero no dentro de las funciones
 - A menos que se declare en la función con la palabra clave **global**
 - O que se acceda con el array **\$GLOBALS[indice]**
- Existen durante todo el tiempo de proceso del fichero
 - Al acabar de procesar la página se eliminan las variables globales
 - Nuevas ejecuciones definen nuevas variables globales

■ Superglobal

- Siempre disponibles
 - Variables predefinidas en el lenguaje, como \$GLOBALS

Ejercicio

■ Variables

- ¿Cuál será la salida del siguiente código?

```
<?php
    $x=1;
    $y=2;
    function prueba() {
        $x=9;
        echo $x.' ';
        echo $GLOBALS['x'].' ';
        global $y;
        $y = $x + $y;
    }
    prueba();
    echo $x.' ';
    echo $y.' ';
?>
```

Variables superglobales

- Variables predefinidas del lenguaje (son arrays asociativos)
 - `$GLOBALS`: Referencia a las variables disponibles en el ámbito global
 - `$_SERVER`: Información del entorno del servidor y de ejecución
 - `$_GET`: Variables que se han pasado al script en un GET
 - `$_POST`: Variables que se han pasado al script en un POST
 - `$_FILES`: Elementos (ficheros) que se han pasado al script en un POST
 - `$_COOKIE`: Variables pasadas al script como cookies HTTP
 - `$_REQUEST`: contiene el contenido de `$_GET`, `$_POST` y `$_COOKIE`
 - `$_ENV`: Variables del entorno del servidor
 - `$_SESSION`: Variables disponibles para la sesión actual

```
echo $_SERVER['SERVER_NAME'];      // nombre del (virtual) host
echo $_SERVER['SERVER_ADDR'];      // IP del servidor
echo $_SERVER['REMOTE_ADDR'];      // IP del cliente
echo $_SERVER['HTTP_USER_AGENT'];  // El navegador del cliente
echo $_SERVER['DOCUMENT_ROOT'];    // Directorio raíz de los docs

echo 'Mi usuario es ' . $_ENV["USER"] . '.';

echo 'Hola ' . htmlspecialchars($_GET["nombre"]); // http://ejemplo.com/?nombre=Juan

echo 'Hola ' . htmlspecialchars($_COOKIE["nombre"]); // si existe el cookie nombre
```

Evitar `register_globals`

- Antiguamente se utilizaba la directiva **`register_globals`** (en `php.ini`) para permitir la inicialización automática de variables globales
 - En PHP 4.2.0 se decidió que fuera desactivada por defecto
 - A partir de PHP 5.4.0 está eliminada
 - <http://www.php.net/manual/es/faq.using.php#faq.register-globals>
- **Agujero de seguridad:**

```
<?php
// Se usa la variable $authorized para indicar si el usuario tiene privilegios
if (authenticated_user()) {
    $authorized = true;
}
// Como no se ha inicializado previamente a false,
// si register_globals está activado, podría inicializarse,
// por ejemplo, desde una petición GET auth.php?authorized=1
// que le daría el valor inicial como true.
// Con código como el siguiente entraría como usuario autorizado:
if ($authorized) {
    include "/datos/muy/privados/listar_datos.php";
}
?>
```

Variables de variables

- Variables que tienen el nombre de una variable
 - Se indican con dos símbolos \$:
 - \$\$variable

```
<?php
$x = 'texto';    // se asigna un string a $x
$$x = 'Hola';    // se usa como una variable de variable
echo $x;         // texto
echo $$x;        // Hola
echo $texto;     // Hola
?>
```

Constantes

- Valores de tipos escalares
- Se declaran con la función **define**
 - `define ('CONSTANTE', 'valor');`
 - El nombre de una constante no puede empezar por \$
 - Normalmente se escriben con mayúsculas
 - Si se intenta redefinir se produce un error de nivel E_NOTICE
- El ámbito de una constante es el script en el que está definida
 - Si se declara en una primera sección de código se puede usar luego

```
<?php
// Constantes
define('AUTOR', 'Juan');
?>
<html>
<head>
    <title>Ejemplo de uso de constantes</title>
</head>
<body>
    <p>Hola <?php echo AUTOR; ?></p>
</body>
</html>
```

Constantes predefinidas

- *"Magical" PHP constants*
 - `__LINE__`
 - Número de la línea de la instrucción que se está ejecutando
 - `__FILE__`
 - Ruta y nombre del fichero
 - Si se usa en un include, se devuelve el nombre del fichero incluido
 - `__DIR__`
 - Directorio del fichero
 - `__FUNCTION__`
 - Nombre de la función
 - `__CLASS__`
 - Nombre de la clase, incluye el namespace en el que está declarada
 - `__TRAIT__`
 - Nombre de un trait (similar a una clase, se verá más adelante)
 - `__METHOD__`
 - Nombre del método
 - `__NAMESPACE__`
 - Espacio de nombres

Constantes predefinidas

- Constantes predefinidas del núcleo
 - `PHP_VERSION` (string)
 - La versión actual de PHP en notación "mayor.menor.edición[extra]"
 - `PHP_DEBUG` (integer)
 - `PHP_OS` (string)
 - `PHP_EOL` (string): símbolo de fin de línea
 - `PHP_INT_MAX` (integer)
 - `E_ERROR` (integer): constante de informe de error
 - `E_WARNING` (integer): indica si se listan los warnings
 - `E_STRICT` (integer): indica si se listan los errores
 - `TRUE`
 - `FALSE`
 - `NULL`
- Lista completa:
<http://www.php.net/manual/es/reserved.constants.php>

Funciones

- Se definen por su nombre y parámetros

```
function nombre($par1 [=valor1], ..., $parn [=valorn]) {  
    // cuerpo de la función  
}
```
- Si un parámetro tiene un valor por defecto, los que le siguen también
- La función se invoca con su nombre y argumentos

```
nombre($arg1, ..., $argn);
```
- El nombre de la función no es sensible a mayúsculas/minúsculas
 - Pero es recomendable llamarlas con el mismo nombre que han sido declaradas
- Las funciones no se pueden sobrecargar
- La función puede devolver un valor con **return**

```
return expresión;
```
- Las funciones se pueden anidar
- Pueden realizarse llamadas recursivas
- PHP ofrece muchas funciones:
<http://www.php.net/manual/es/funcref.php>

Funciones

- Los parámetros se pueden pasar
 - Por valor
 - Se hace una copia del argumento que se pasa a la función
 - Los cambios realizados no son visibles fuera de la función
 - Por referencia (indicando **&** delante del parámetro)

```
function nombre(&$par1) { /*...*/ }
```

 - En este caso los cambios al parámetro sí afectan a la variable que se pasa por referencia
- Parámetros por defecto (similar a C++)
 - Tiene que ser una expresión que dé un valor constante
 - Los argumentos con valores por defecto se tienen que poner al final

```
function nombre($par1, $par2=valor_constante) { /*...*/ }
```
- Lista de argumentos variable
 - **func_num_args()**: número de argumentos que se han pasado
 - **func_get_arg(n)**: n-ésimo argumento que se ha pasado a la función
 - Si $n > \text{func_num_args}()$, devuelve false
 - **func_get_args()**: array de parámetros que se han pasado a la función

Tipos básicos

- **Escalares**
 - boolean
 - integer
 - float (floating-point number, aka double)
 - string
- **Tipos compuestos**
 - array
 - object
- **Tipos especiales**
 - resource
 - NULL

Tipos escalares

- **Boolean**
 - FALSE, false: 0, 0.0, "", "0", array de 0 elementos, NULL y variables sin inicializar
 - TRUE, true: cualquier otro valor
- **Integer**
 - Representados en base 10 (decimal), 8 (octal) o 16 (hexadecimal)
 - En las versiones recientes se guardan con 64 bits
 - Si al evaluar una expresión sobrepasa el valor máximo (PHP_INT_MAX), se convierte a float
- **Float**
 - Números reales (no hay diferencia entre float y double)

```
$usuario = "Juan";  
$activo = true;  
$activo = 1; // true  
$octal = 0623;  
$hexadecimal = 0xF4;  
$cuenta = 33;  
$saldo = 4534.32;  
$saldo = 4.53432e3
```

Tipos escalares

■ String

- Cadenas de caracteres ASCII, entre comillas simples o dobles
- Para trabajar con Unicode se usan las funciones
 - `utf8_encode()`
 - `utf8_decode()`
- Se puede acceder a los caracteres de un String como en un array `$cadena{n}`
 - El primer carácter es el del índice 0
- string **substr** (string \$string , int \$start [, int \$length])
 - Devuelve un string desde la posición \$start y longitud \$length
 - Si \$start es negativo cuenta hacia atrás desde la última posición
 - Si la longitud del string es menor o igual que \$start, devuelve FALSE
 - Si no se especifica \$length se considera el resto del string
 - Si \$length es negativo se restan esos caracteres del final

```
$texto = "Bienvenido";  
echo $texto{3}; // n  
echo $texto{0}; // B  
echo substr($texto, 0, 4); // Bien  
echo substr($texto, -4); // nido
```

Tipos escalares

■ String (con comillas dobles)

- Sustitución de variables en strings
 - Al encontrar una variable (un \$) se sustituye por su valor
 - Se puede encerrar la variable o su nombre con { }
- Interpretación de secuencias de escape
 - `\n` Salto de línea `\r` Retorno de carro `\t` Tabulador horizontal
 - `\\` Barra `\$` Signo de dólar
 - `\"` Comillas dobles `\'` Comilla simple
 - `\[0-7]{1,3}` Número en notación octal
 - `\x[0-9A-Fa-f]{1,2}` Número en notación hexadecimal

```
$texto = "Bienvenido";  
$nombre = "Juan";  
echo "$texto, ${nombre}."; // Bienvenido, Juan.
```

Tipos escalares

■ String (con comillas simples)

- No sustituyen las variables por su valor
- Ni siguen las secuencias de escape
 - Solo para especificar una comilla simple se especifica con /'

```
echo 'String de comillas simples'; // String de comillas simples

echo 'Se puede
poner con
varias líneas'; // En el HTML los saltos de línea no se interpretarán, pero están

echo 'Juan said: "I\'ll be back"'; // Juan said: "I'll be back"

echo 'Ficheros C:\\xampp\\htdocs\\*.html '; // Ficheros C:\\xampp\\htdocs\\*.html

echo 'Ficheros C:\\xampp\\htdocs\\*.html'; // Ficheros C:\\xampp\\htdocs\\*.html

echo 'No salta línea con \\n. Se imprime igual que \\n';
    // No salta línea con \\n. Se imprime igual que \\n

echo 'Las $variables no se $interpretan.'; // Las $variables no se $interpretan.
```

Tipos escalares

■ String

- Operador de concatenación . (punto)

```
$texto = "Bienvenido";
$nombre = "Juan";
echo $texto.", ".$nombre." ".PHP_EOL; // Bienvenido, Juan.
```

- Asignación y concatenación .=

```
$texto = "Bienvenido";
$texto .= ", Juan.";
echo $texto; // Bienvenido, Juan.
```

- Más funciones sobre strings:


<http://www.php.net/manual/en/ref.strings.php>

Array

- Arrays asociativos: secuencias de pares (clave, valor)
 - Clave: entero o string
 - Los strings que representen números se convierten en enteros
 - Float y boolean se convierten en enteros (true→1, false→0)
 - Valor: cualquier cosa
- Se crean con la función **array** y una secuencia de pares **clave=>valor** separados por comas
array(clave=> valor, clave2 => valor2, clave3 => valor3, ...)
 - Desde PHP 5.4 se pueden usar corchetes [] en vez de paréntesis ()

```
$array = array(  
    "uno" => 1,  
    "dos" => 2,  
);  
  
// desde PHP 5.4  
$array = [  
    "uno" => 1,  
    "dos" => 2,  
];
```

```
$array = array(  
    1 => "a",  
    "1" => "b",  
    1.5 => "c",  
    true => "d",  
);  
var_dump($array);
```




```
array(1) {  
    [1]=> string(1) "d"  
}
```

Array

- Si no se usa clave, se indexan 0, 1, 2, ...
 - Si se añade un elemento con un número específico como clave, la siguiente clave será el siguiente número
- Si hubiera claves string, el primer entero que se usaría al no especificar clave sería el 0
- **array_values()** reindexa un array (pone todos los elementos ordenados desde 0, suprimiendo aquellos que no tuvieran valor)

```
$array = array("uno", "dos", 5=>"tres", "cuatro");  
var_dump($array);
```




```
array(4) {  
    [0]=> string(3) "uno"  
    [1]=> string(3) "dos"  
    [5]=> string(4) "tres"  
    [6]=> string(6) "cuatro"  
}
```

Array

- Acceso a los elementos: **array[clave]** o **array{clave}**
- La clave se especifica entre comillas, a menos que se trate de una constante o una variable

```
$array = array(  
    "uno" => "hola",  
    42    => 24,  
    "multi" => array(  
        "dimensional" => array(  
            "array" => "bravo"  
        )  
    )  
);  
  
var_dump($array["uno"]);  
var_dump($array[42]);  
var_dump($array["multi"]["dimensional"]["array"]);
```



```
string(4) "hola"  
int(24)  
string(5) "bravo"
```

Array

- Modificación de los elementos de un array
\$arr[clave] = valor;
\$arr[] = valor;
- Si \$arr no existe, se creará
- Para eliminar un par (clave, valor), utilizar la función **unset()**
 - Los índices (claves) no se reutilizan

```
$arr = array(1, 4=> "cuatro");  
  
$arr[] = 22; // $arr[5] = 22;  
  
$arr["x"] = 33; // Nuevo elemento con clave "x"  
  
unset($arr[5]); // Elimina el elemento del array  
  
unset($arr); // Borra todo el array
```

Array

- La asignación de arrays implica una copia
 - Si se quiere copiar una referencia de array, usar el operador de referencia

```
$arr1 = array(2, 3);  
$arr2 = $arr1;  
$arr2[] = 4; // se cambia $arr2 pero no $arr1  
  
$arr3 = &$arr1;  
$arr3[] = 4; // se cambian $arr3 y $arr1
```

Array

- Arrays multidimensionales: arrays de arrays

```
$frutas = array ( "frutas" => array ( "a" => "albaricoque",  
                                     "c" => "coco",  
                                     "m" => "manzana",  
                                     "n" => "naranja"  
                                   ),  
                "numeros" => array ( 1,  
                                     2,  
                                     3,  
                                     4,  
                                     5,  
                                   ),  
                );  
  
// Ejemplos de uso:  
echo $frutas["frutas"]["c"]; // "coco"  
  
echo $frutas["numeros"][3]; // 4  
  
unset($frutas["frutas"]["a"]); // elimina "albaricoque"  
  
echo $frutas["frutas"]["a"]; // Notice: Undefined index
```

Array

■ Recorrer un array: **foreach**

`foreach (array as $valor)
sentencias`

- Recorre el array
- En cada iteración, el valor del elemento actual se asigna a *\$valor* y el puntero interno del array avanza una posición

`foreach (array as $clave => $valor)
sentencias`

- Similar al anterior y asigna la clave del elemento actual a la variable *\$clave* en cada iteración

```
<?php
$arr = array("uno", "dos", "tres");

foreach ($arr as $valor)
    echo "$valor <br />";

foreach ($arr as $clave => $valor) {
    echo "$clave => $valor <br />";
}
?>
```

Ejercicio

■ ¿Qué se obtendrá con el siguiente código?

```
<?php
$array = array(1, 2, 3, 4, 5);
print_r($array);

foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

$array[] = 6;
print_r($array);

$array = array_values($array); // re-indexa el array

$array[] = 7;
print_r($array);
?>
```

Ejemplo adaptado del Manual de PHP: <http://www.php.net/manual/es/language.types.array.php>

Ejercicio

- Haz un programa PHP que genere la tabla de multiplicar entre un número mínimo y uno máximo
 - Por ejemplo, entre 1 y 12:

La tabla de multiplicar

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	4	6	8	10	12	14	16	18	20	22	24
3	3	6	9	12	15	18	21	24	27	30	33	36
4	4	8	12	16	20	24	28	32	36	40	44	48
5	5	10	15	20	25	30	35	40	45	50	55	60
6	6	12	18	24	30	36	42	48	54	60	66	72
7	7	14	21	28	35	42	49	56	63	70	77	84
8	8	16	24	32	40	48	56	64	72	80	88	96
9	9	18	27	36	45	54	63	72	81	90	99	108
10	10	20	30	40	50	60	70	80	90	100	110	120
11	11	22	33	44	55	66	77	88	99	110	121	132
12	12	24	36	48	60	72	84	96	108	120	132	144

Objetos

- Desde PHP 5 se ha mejorado el modelo de objetos
- Se pueden definir
 - Clases
 - Traits
 - Interfaces
- Los objetos se crean con el operador **new**
- Las variables de objetos son referencias a los objetos (no copias de los objetos)
 - Las asignaciones de variables de referencia a objetos copian referencias, no los objetos
 - Si se quiere una copia del objeto hay que *clonar* el objeto:
`$copia_de_objeto = clone $objeto;`

Clases

- Se definen con **class**
- Seguida del nombre de clase
 - Una etiqueta válida PHP que no sea una palabra reservada
- Seguido de las
 - Propiedades
 - Constantes
 - Variables
 - Pueden ser de clase declarándolas como **static**
 - Métodos
 - Funciones
 - Constructor `__construct()`
 - Destructor `__destruct()`
 - PHP no tiene sobrecarga de métodos: no puede haber varias versiones de un método con distintos parámetros
 - Sería complicado por ser un lenguaje débilmente tipado

Herencia de clases

- **extends**
 - Herencia simple (una clase solo puede heredar de una)
 - Se pueden sobrescribir los métodos y propiedades que no estén declarados como **final** en la superclase
 - Como no hay sobrecarga de métodos en PHP tampoco se puede sobrescribir un método con distintos parámetros que la superclase
 - Se puede acceder a los métodos de la superclase con `parent::`
`parent::metodo();`

Interfaces

- Definen conjuntos de métodos públicos
- Tienen que ser implementados por clases
 - Se declaran las interfaces que implementa con la palabra **implements**
 - Una clase puede implementar más de una interface

```
interface Bolsa {  
    public function compra();  
    public function venta();  
}  
  
class BolsaDeMadrid implements Bolsa {  
    // implementación de los métodos compra() y venta()  
}
```

Clases abstractas

- Clases que no se pueden instanciar, solo heredar
- Pueden tener métodos sin implementar (abstractos)

```
abstract class ClaseAbstracta {  
    public function metodo() {  
        // implementación del método  
    }  
    public abstract function abstracta();  
}  
  
class ClaseConcreta extends ClaseAbstracta {  
    // implementación de la función abstracta()  
}
```

Traits

- Un trait permite agrupar funciones
- Mecanismo para hacer una especie de herencia múltiple
 - Poco recomendable, hay alternativas que permiten mejores diseños

```
trait A {  
    public function printa() {  
        echo 'a';  
    }  
}  
  
trait B {  
    public function printb() {  
        echo 'b';  
    }  
}  
  
class AB {  
    use A, B;  
}  
  
$o = new AB();  
$o->printa();  
$o->printb();
```

Acceso a los miembros de una clase

- Con el operador de objeto (->)
 - *\$objeto->propiedad*
 - *\$objeto->metodo()*
- Al declararse se define la **visibilidad** de los miembros:
 - **public** – visible también fuera de la clase
 - **protected** – visible también en subclases
 - **private** – visible solo dentro de la clase

Objetos

- Creación de objetos: **new**
 - Normalmente se especifica el nombre de la clase correspondiente
 - La clase tiene que estar definida antes de la instanciación
`$instancia = new Clase();`
- También se puede hacer con una variable
`$clase = 'Clase';`
`$instancia = new $clase(); // new Clase()`

Objetos

- Pseudo-variable **\$this**
 - Referencia al propio objeto
 - Si el objeto no está definido saldrá un error si E_STRICT está activo

```
<?php
class Prueba
{
    function listaClase() {
        if (isset($this)) {
            echo '$this es un objeto de la clase: ';
            echo get_class($this);
        }
        else
            echo '$this NO est&aacute; definido.';
    }
}

$p = new Prueba();
$p->listaClase();

Prueba::listaClase();
?>
```

Objetos

- Operador de resolución de ámbito ::
 - Permite el acceso a propiedades y métodos sobreescritos en una clase
- Usando pseudo-variables
 - **\$parent**
 - Referencia a la superclase
 - Se puede invocar un método de la superclase como `parent::metodo();`
 - **\$self**
 - Referencia a la clase (diferente de `$this` que referencia a un objeto)
 - Puede usarse para referirse a métodos o propiedades estáticas `self::$variable_static;`

Ejercicio

- Implementa una clase `CuentaBancaria` en PHP que implemente la interfaz `Cuenta`

```
interface Cuenta {  
    public function ingreso($cantidad);  
    public function reintegro($cantidad);  
}
```

- ¿Qué excepciones podrían generar estos métodos?
- Implementa otra clase `CuentaBancariaPreferente` que pueda tener asociada un crédito de una cantidad que se define al abrirla y con un método que permita modificarla

```
interface Credito {  
    public function creditoMax();  
    public function cambiarCreditoMax($cantidad);  
    public function solicitaCredito($cantidad);  
    public function devuelveCredito($cantidad);  
}
```

Resource

- Variable que tiene una referencia a un recurso externo
- Utilizados por funciones específicas
- Tipos de recursos y funciones asociadas
 - <http://www.php.net/manual/en/resource.php>
- Ejemplos
 - ftp
 - Se crea con `ftp_open()`
 - Se usa con `ftp_login()`, `ftp_mkdir()`, `ftp_get()`, `ftp_chdir()`, etc.
 - Se elimina con `ftp_close()`
 - imap
 - ldap link
 - mysql query
 - pdf document
 - xml

NULL

- Representa una variable sin valor: NULL o null
- Una variable se considera null si
 - Ha sido asignada la constante NULL
 - No se le ha asignado ningún valor
 - Se le ha aplicado la función `unset()`
- `bool is_null (mixed $var)`
- `void unset (mixed $var [, mixed $...])`
 - Destruye las variables especificadas
 - Si se llama dentro de una función para una variable global solo se destruye esa variable dentro de esa función (a menos que se especifique con `GLOBALS`)

```
function destruye() {  
    global $x;  
    unset($x);  
}  
$x="equis";  
destruye();  
echo $x;
```

```
function destruye() {  
    global $x;  
    unset($GLOBALS['x']);  
}  
$x="equis";  
destruye();  
echo $x;
```

Conversión de tipos (casting)

- PHP es un lenguaje débilmente tipado
 - No hay que declarar el tipo de las variables, que se deduce
 - PHP hace una conversión automática de tipos según sea necesario
- Es posible, sin embargo, hacerlo explícitamente con los operadores de casting:
 - (array)
 - (bool) o (boolean)
 - (int) o (integer)
 - (object)
 - (real) o (double) o (float)
 - (string)

```
$precio = (double)20;    // 20.0
$vueltas = (int)13.4;    // 13
```

Conversión de tipos (casting)

- Conversiones a integer
 - boolean → integer: FALSE → 0, TRUE → 1
 - float → integer: Se redondea el valor hacia abajo
 - Si el valor sobrepasa el valor máximo el resultado es indefinido
 - string → integer: Se interpreta el número representado
 - Otro valor: primero se convierte a boolean y después se realiza la conversión
- Conversiones a string
 - boolean → string: FALSE → "" (cadena vacía), TRUE → "1"
 - integer/float → string: Cadena que representa el número
 - array → string: Cadena "Array"
 - object → string: Cadena "Object".
 - NULL → string: "" (cadena vacía)

Funciones sobre tipos con variables

- string **gettype**(mixed \$variable)
- bool **settype** (mixed &\$var , string \$type)
 - Devuelve y pone el tipo de la variable como un string
 - "boolean"
 - "integer"
 - "double" (por razones históricas, en vez de "float")
 - "string"
 - "array"
 - "object"
 - "resource"
 - "NULL"
 - "unknown type"
- bool **is_***(mixed \$variable)
 - is_array, is_bool, is_double, is_float, is_int, is_integer, is_long, is_null, is_numeric, is_object, is_real, is_resource, is_scalar, is_string

Funciones sobre variables

- void **var_dump**(mixed \$variable)
 - Muestra información (tipo, valor) de la variable
- **print_r**(mixed \$variable)
 - Imprime información legible sobre una variable
- bool **empty** (mixed \$var)
 - TRUE si la variable no existe o vale FALSE
- bool **isset** (mixed \$var [, mixed \$...])
 - Determina si la variable tiene un valor y no es NULL
- string **strval** (mixed \$var)
 - Obtiene el valor de la variable como string
- int **intval** (mixed \$var [, int \$base = 10])
 - Obtiene el valor de la variable como entero
- float **floatval** (mixed \$variable)
 - Obtiene el valor float de la variable

Funciones sobre variables

- Serialización
 - string **serialize** (mixed \$value)
 - Genera una representación almacenable de un valor
 - Se regenera con **unserialize** (string \$str)

Expresiones

- Asignación
 - Guarda un valor específico en una variable
`$variable = expresión;`
 - Asignación por referencia (&): las dos variables se refieren al mismo espacio de memoria
`$variable = &$otra_variable;`
- Expresiones numéricas
 - Operadores aritméticos:
 - +, ++, -, --, *, /, % (módulo)
 - Asignaciones con operadores aritméticos
 - +=, -=, *=, /=, %=

Expresiones

- Expresiones lógicas
 - Operadores lógicos
 - && and
 - || or
 - xor
 - !
 - Precedencia: !, &&, ||, and, xor, or
- Operador ternario
 - expresión ? afirmativa : negativa

`$maximo = $primero > $segundo ? $primero : $segundo`

Expresiones

- Expresiones de comparación
 - Operadores relacionales: ==, !=, >, <, >=, <=, ===, !==
 - Conversión automática de tipos en las comparaciones
 - PHP realiza conversiones automáticas entre tipos para llevar a cabo la comparación cuando sea necesario
 - Si un operando es una cadena y el otro un número, se intenta convertir la cadena a número. Si no se puede convertir la comparación devuelve false
 - Si uno de los operandos es un booleano y el otro un número se convierte el booleano a número (true 1, false 0)
 - Comparación estricta (===, !==): no se realiza conversión alguna, tienen que ser iguales en valor y tipos

Control de flujo

■ Instrucciones condicionales

■ if

```
if (condición) {  
    // Instrucciones  
}  
elseif (condición) {  
    // Instrucciones  
}  
// otros elseif ...  
else {  
    // Instrucciones  
}
```

■ Sintaxis alternativa (varios bloques):

```
<?php if (condición) : ?>  
Texto (HTML, JavaScript, PHP)  
<?php else : ?>  
Más texto  
<?php endif; ?>
```

```
<p>  
<?php if ($nombre==null) : ?>  
    Hola.  
<?php else : ?>  
    Hola <?php echo $nombre ?>.  
<?php endif; ?>  
</p>
```

Control de flujo

■ Instrucciones condicionales

■ switch

```
switch (expresión) { // la expresión debe dar un tipo escalar  
case valor1:  
    // Instrucciones caso 1  
    break; // para acabar el switch  
case valor2:  
    // Instrucciones caso 2  
    break;  
// otros case ...  
default: // opcional  
    // Instrucciones si no se diera ningún caso  
}
```

■ También se puede poner en varios bloques

- El primer case tiene que ir en el mismo bloque PHP de switch

Control de flujo

■ Bucles

■ while

```
while( condición ){  
    // Instrucciones  
}
```

• Ejemplo:

```
while( true ) {  
    bucle_infinito();  
}
```

■ do while

```
do {  
    // Instrucciones  
} while( condición )
```

```
<?php while (condición) : ?>  
Texto (HTML, JavaScript, PHP)  
<?php endwhile; ?>
```

■ También tienen la posibilidad de hacerse en varios bloques

■ Sentencias para control de bucles

■ Salir del bucle **break**

■ Saltar a la siguiente iteración **continue**

Control de flujo

■ Bucles

■ for

```
for ( inicialización; condición; actualización) {  
    // Instrucciones  
}
```

• Ejemplo:

```
for ($i=0; $i<$limite; $i++) {  
    procesa($i);  
}
```

■ Sintaxis alternativa (varios bloques):

```
<?php for (expr1; expr2; expr3): ?>  
Texto (HTML, JavaScript, PHP)  
<?php endfor; ?>
```