



EBook Gratis

APRENDIZAJE PHP

Free unaffiliated eBook created from
Stack Overflow contributors.

#php

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con PHP.....	2
Observaciones.....	2
Versiones.....	3
PHP 7.x.....	3
PHP 5.x.....	3
PHP 4.x.....	3
Versiones heredadas.....	4
Examples.....	4
Salida HTML del servidor web.....	4
Salida no HTML desde servidor web.....	5
¡Hola Mundo!.....	6
Separación de instrucciones.....	7
PHP CLI.....	8
Disparando.....	8
Salida.....	8
Entrada.....	9
Servidor incorporado de PHP.....	9
Ejemplo de uso.....	9
Configuración.....	10
Troncos.....	10
Etiquetas PHP.....	10
Etiquetas estándar.....	10
Etiquetas de eco.....	10
Etiquetas cortas.....	11
Etiquetas ASP.....	11
Capítulo 2: Actuación.....	12
Examples.....	12
Perfilando con XHProf.....	12

Uso de memoria.....	12
Perfilando con Xdebug	13
Capítulo 3: Alcance variable.....	17
Introducción.....	17
Examples.....	17
Variables globales definidas por el usuario.....	17
Variables superglobales.....	18
Propiedades estáticas y variables.....	18
Capítulo 4: Análisis de cuerdas.....	20
Observaciones.....	20
Examples.....	20
Dividir una cadena por separadores.....	20
Buscando una subcadena con strpos.....	21
Comprobando si existe una subcadena.....	21
Búsqueda a partir de un offset	21
Consigue todas las apariciones de una subcadena.....	22
Analizando la cadena usando expresiones regulares.....	22
Subcadena.....	23
Capítulo 5: Análisis de HTML.....	25
Examples.....	25
Analizar HTML desde una cadena.....	25
Utilizando XPath.....	25
SimpleXML	25
Presentación.....	25
Análisis de XML utilizando un enfoque de procedimiento.....	26
Analizar XML utilizando el enfoque OOP.....	26
Accediendo a los niños y atributos.....	26
Cuando conoces sus nombres:.....	26
Cuando no sabes sus nombres (o no quieres saberlos):.....	27
Capítulo 6: APCu.....	28
Introducción.....	28

Examples	28
Almacenamiento y recuperación simples	28
Almacenar información	28
Iterando sobre las entradas	28
Capítulo 7: Aprendizaje automático	30
Observaciones	30
Examples	30
Clasificación utilizando PHP-ML	30
SVC (Clasificación de vectores de soporte)	30
k-vecinos más cercanos	31
Clasificador NaiveBayes	31
Caso practico	32
Regresión	32
Regresión de vectores de apoyo	32
Regresión lineal de LeastSquares	33
Caso practico	34
Agrupación	34
k-medios	34
DBSCAN	34
Caso practico	35
Capítulo 8: Arrays	36
Introducción	36
Sintaxis	36
Parámetros	36
Observaciones	36
Ver también	36
Examples	36
Inicializando una matriz	37
Comprobar si existe la clave	39
Comprobando si existe un valor en la matriz	40
Validando el tipo de matriz	41

Interfaces ArrayAccess e Iterador.....	41
Creando una matriz de variables.....	45
Capítulo 9: Asegurate recuerdame.....	46
Introducción.....	46
Examples.....	46
"Mantenerme conectado" - el mejor enfoque.....	46
Capítulo 10: Autenticación HTTP.....	47
Introducción.....	47
Examples.....	47
Autenticación simple.....	47
Capítulo 11: BC Math (calculadora binaria).....	48
Introducción.....	48
Sintaxis.....	48
Parámetros.....	48
Observaciones.....	50
Examples.....	50
Comparación entre BCMath y operaciones aritméticas flotantes.....	50
bcadd vs float + float.....	50
bcsub vs float-float.....	50
bcmul vs int * int.....	50
bcmul vs float * float.....	50
bcddiv vs float / float.....	51
Uso de bcmath para leer / escribir un binario largo en un sistema de 32 bits.....	51
Capítulo 12: Bucles.....	53
Introducción.....	53
Sintaxis.....	53
Observaciones.....	53
Examples.....	53
para.....	53
para cada.....	54
descanso.....	55

hacer ... mientras	56
continuar.....	56
mientras	58
Capítulo 13: Buffer de salida	59
Parámetros.....	59
Examples.....	59
Uso básico obteniendo contenido entre buffers y clearing	59
Buffers de salida anidados.....	60
Capturando el buffer de salida para reutilizarlo más tarde.....	61
Ejecutando buffer de salida antes de cualquier contenido.....	62
Uso del búfer de salida para almacenar contenidos en un archivo, útil para informes, factu.....	62
Procesando el búfer a través de una devolución de llamada	63
Transmitir salida al cliente.....	64
Uso típico y razones para usar ob_start	64
Capítulo 14: Cache	65
Observaciones.....	65
Instalación.....	65
Examples.....	65
Caché utilizando memcache.....	65
Almacenamiento de datos	66
Obtener datos	66
Borrar datos	66
Pequeño escenario para el almacenamiento en caché.....	66
Caché utilizando caché APC	67
Capítulo 15: Cierre	68
Examples	68
Uso básico de un cierre.....	68
Utilizando variables externas.....	69
Encuadernación de cierre básico.....	69
Cierre de encuadernación y alcance.....	70
Encuadernación de un cierre para una llamada.....	71
Utilizar cierres para implementar patrón observador.....	72

Capítulo 16: Clase de fecha y hora	74
Examples	74
getTimestamp	74
Establece la fecha	74
Aregar o restar intervalos de fecha	74
Crea DateTime desde un formato personalizado	75
Imprimir DateTimes	75
Formato	75
Uso	76
Procesal	76
Orientado a objetos	76
Equivalente de procedimiento	76
Cree una versión inmutable de DateTime desde Mutable antes de PHP 5.6	76
Capítulo 17: Clases y objetos	78
Introducción	78
Sintaxis	78
Observaciones	78
Clases y componentes de interfaz	78
Examples	79
Interfaces	79
Introducción	79
Realización	79
Herencia	80
Ejemplos	81
Constantes de clase	82
definir constantes de clase vs	84
Usando :: class para recuperar el nombre de la clase	85
Enlace estático tardío	85
Clases abstractas	86
Nota IMPORTANTE	88
Separación de nombres y carga automática	89

Vinculación dinámica	90
Método y visibilidad de la propiedad	91
Público	91
Protegido	92
Privado	93
Llamar a un constructor padre al crear una instancia de un hijo	94
Palabra clave final	94
\$ esto, auto y estático más el singleton	96
El singleton	98
Autocarga	98
Clases anónimas	100
Definiendo una clase básica	101
Constructor	102
Extendiendo otra clase	102
Capítulo 18: Cliente de jabón	104
Sintaxis	104
Parámetros	104
Observaciones	104
Examples	106
Modo WSDL	106
Modo no WSDL	107
Mapas de clase	107
Rastreo de solicitud y respuesta SOAP	108
Capítulo 19: Comentarios	109
Observaciones	109
Examples	109
Comentarios de una sola línea	109
Comentarios multilínea	109
Capítulo 20: Cómo desglosar una URL	110
Introducción	110
Examples	110

Usando parse_url ().....	110
Utilizando explotar ().....	111
Usando basename ().....	112
Capítulo 21: Cómo detectar la dirección IP del cliente.....	113
Examples.....	113
Uso correcto de HTTP_X_FORWARDED_FOR.....	113
Capítulo 22: Compilar extensiones de PHP.....	115
Examples.....	115
Compilando en linux.....	115
Pasos para compilar.....	115
Cargando la extensión en PHP.....	116
Capítulo 23: Constantes.....	117
Sintaxis.....	117
Observaciones.....	117
Examples.....	117
Comprobando si se define constante.....	117
Simple cheque.....	117
Obteniendo todas las constantes definidas.....	118
Definiendo constantes.....	118
Definir constante utilizando valores explícitos.....	119
Definir constante utilizando otra constante.....	119
Constantes reservadas.....	119
Condicional define.....	119
const vs define.....	120
Constantes de clase.....	120
Matrices constantes.....	121
Ejemplo de clase constante.....	121
Ejemplo de constante llana.....	121
Usando constantes.....	121
Capítulo 24: Constantes mágicas.....	123
Observaciones.....	123

Examples.....	123
Diferencia entre __FUNCTION__ y __METHOD__.....	123
Diferencia entre __CLASS__, get_class () y get_called_class ().....	124
Constantes de archivo y directorio.....	124
Archivo actual.....	124
Directorio actual.....	125
Separadores.....	125
Capítulo 25: Contribuyendo al Manual de PHP.....	127
Introducción.....	127
Observaciones.....	127
Examples.....	127
Mejorar la documentación oficial.....	127
Consejos para contribuir al manual.....	127
Capítulo 26: Contribuyendo al PHP Core.....	129
Observaciones.....	129
Contribuyendo con la corrección de errores.....	129
Contribuyendo con adiciones de características.....	129
Lanzamientos.....	130
Versiones.....	130
Examples.....	130
Configuración de un entorno de desarrollo básico.....	131
Capítulo 27: Convenciones de codificación.....	132
Examples.....	132
Etiquetas PHP.....	132
Capítulo 28: Corrientes.....	133
Sintaxis.....	133
Parámetros.....	133
Observaciones.....	133
Examples.....	134
Registro de una envoltura de flujo.....	134
Capítulo 29: Crea archivos PDF en PHP.....	136

Examples.....	136
Empezando con PDFlib.....	136
Capítulo 30: Criptografía.....	137
Observaciones.....	137
Examples.....	137
Cifrado simétrico.....	137
Cifrado.....	137
Descifrado.....	137
Base64 Codifica y Decodifica.....	138
Cifrado y descifrado simétricos de archivos grandes con OpenSSL.....	138
Cifrar archivos.....	138
Descifrar archivos.....	139
Cómo utilizar.....	140
Capítulo 31: Datos de solicitud de lectura.....	141
Observaciones.....	141
Elegir entre GET y POST.....	141
Solicitar vulnerabilidades de datos.....	141
Examples.....	141
Manejo de errores de carga de archivos.....	141
Lectura de datos POST.....	142
Leyendo datos GET.....	142
Lectura de datos POST sin procesar.....	143
Subiendo archivos con HTTP PUT.....	143
Pasando matrices por POST.....	144
Capítulo 32: Depuración.....	146
Examples.....	146
Variables de dumping.....	146
Mostrando errores.....	146
phpinfo ().....	147
Advertencia.....	147
Introducción.....	147

Ejemplo	148
Xdebug	148
phpversion ()	149
Introducción	149
Ejemplo	149
Informe de errores (utilícelos ambos)	149
Capítulo 33: Despliegue de Docker	150
Introducción	150
Observaciones	150
Examples	150
Obtener imagen docker para php	150
Escritura dockerfile	150
Ignorando archivos	151
Imagen del edificio	151
Iniciar contenedor de aplicaciones	151
Contenedor de control	151
Registros de aplicación	151
Capítulo 34: DOP	153
Introducción	153
Sintaxis	153
Observaciones	153
Examples	153
Conexión y recuperación de PDO básica	153
Prevención de la inyección SQL con consultas parametrizadas	154
DOP: conexión a servidor MySQL / MariaDB	156
Conexión estándar (TCP / IP)	156
Conexión de zócalo	156
Transacciones de base de datos con DOP	157
DOP: obtener el número de filas afectadas por una consulta	160
DOP :: lastInsertId ()	160
Capítulo 35: Ejecutando sobre una matriz	162

Examples	162
Aplicando una función a cada elemento de una matriz.....	162
Dividir matriz en trozos.....	163
Implementando una matriz en una cadena.....	164
array_reduce.....	164
Arreglos de "destrucción" usando list ().....	166
Presionar un valor en una matriz.....	166
Capítulo 36: Enchufes.....	168
Examples.....	168
Socket cliente TCP.....	168
Creando un socket que usa el TCP (Protocolo de Control de Transmisión).....	168
Conecte el zócalo a una dirección especificada.....	168
Enviando datos al servidor.....	168
Recibiendo datos del servidor.....	168
Cerrando el zócalo.....	169
Zócalo del servidor TCP.....	169
Creación de zócalo.....	169
Enlace de zócalo.....	169
Establecer un zócalo para escuchar.....	170
Manejo de conexión.....	170
Cerrando el servidor.....	170
Manejo de errores de socket	170
Servidor UDP socket	171
Creando un socket de servidor UDP.....	171
Atar un socket a una dirección.....	171
Enviando un paquete.....	171
Recibiendo un paquete.....	171
Cerrando el servidor.....	171
Capítulo 37: Envíando email.....	173
Parámetros.....	173

Observaciones.....	173
Examples.....	174
Envío de correo electrónico: conceptos básicos, más detalles y un ejemplo completo.....	174
Enviando correo electrónico HTML usando correo ().....	177
Enviar correo electrónico de texto sin formato utilizando PHPMailer.....	177
Enviar correo electrónico con un archivo adjunto utilizando correo ().....	178
Codificaciones de transferencia de contenido.....	179
Envío de correo electrónico HTML utilizando PHPMailer.....	180
Envío de correo electrónico con un archivo adjunto utilizando PHPMailer.....	180
Enviar correo electrónico de texto sin formato utilizando Sendgrid.....	181
Enviar correo electrónico con un archivo adjunto utilizando Sendgrid.....	182
Capítulo 38: Errores comunes.....	183
Examples.....	183
\$ Inesperado fin.....	183
Llame a fetch_assoc en boolean.....	183
Capítulo 39: Espacios de nombres.....	185
Observaciones.....	185
Examples.....	185
Declarando espacios de nombres.....	185
Hacer referencia a una clase o función en un espacio de nombres.....	186
¿Qué son los espacios de nombres?.....	187
Declarar sub-espacios de nombres.....	187
Capítulo 40: Estructuras de Control.....	189
Examples.....	189
Sintaxis alternativa para estructuras de control.....	189
mientras.....	189
hacer mientras.....	189
ir.....	190
declarar.....	190
si mas.....	190
incluir y requerir.....	191
exigir.....	191

incluir	191
regreso.....	192
para	193
para cada.....	193
si otra cosa más.....	194
Si.....	194
cambiar.....	194
Capítulo 41: Estructuras de datos SPL	196
Examples.....	196
SplFixedArray.....	196
Diferencia de PHP Array	196
Creando la matriz	198
Cambiar el tamaño de la matriz	198
Importar a SplFixedArray y exportar desde SplFixedArray	199
Capítulo 42: Examen de la unidad	201
Sintaxis.....	201
Observaciones.....	201
Examples.....	201
Pruebas de reglas de clase.....	201
PHPUnit Data Providers.....	204
Array de matrices.....	205
Iteradores.....	206
Generadores.....	207
Excepciones de prueba.....	208
Capítulo 43: Expresiones regulares (regexp / PCRE)	210
Sintaxis.....	210
Parámetros.....	210
Observaciones.....	210
Examples.....	210
Coincidencia de cadenas con expresiones regulares.....	211
Dividir cadena en matriz por una expresión regular.....	211

Cadena sustituyendo con expresión regular.....	212
Partido RegExp global.....	212
Cadena reemplazar con devolución de llamada.....	214
Capítulo 44: Extensión de roscado múltiple.....	215
Observaciones.....	215
Examples.....	215
Empezando.....	215
Uso de piscinas y trabajadores.....	216
Capítulo 45: Filtros y funciones de filtro.....	218
Introducción.....	218
Sintaxis.....	218
Parámetros.....	218
Examples.....	218
Validar correo electrónico.....	218
Validar un valor es un entero.....	219
Validando un número entero cae en un rango.....	220
Validar una URL.....	220
Desinfectar filtros.....	222
Validación de valores booleanos.....	223
Validar un número es un flotador.....	223
Validar una dirección MAC.....	224
Sanitze Direcciones de correo electrónico.....	224
Desinfectar enteros.....	225
Desinfectar URL.....	225
Desinfectar flotadores.....	226
Validar direcciones IP.....	227
Capítulo 46: Formato de cadena.....	230
Examples.....	230
Extracción / sustitución de subcadenas.....	230
Interpolación de cuerdas.....	230
Capítulo 47: Funciones.....	233
Sintaxis.....	233

Examples	233
Uso básico de la función	233
Parámetros opcionales	233
Pasando Argumentos por Referencia	234
Listas de argumentos de longitud variable	235
Alcance de la función	237
Capítulo 48: Funciones de hash de contraseña.....	238
Introducción	238
Sintaxis	238
Observaciones	238
Selección de algoritmo	238
Algoritmos seguros	238
Algoritmos inseguros	238
Examples	239
Determine si un hash de contraseña existente puede actualizarse a un algoritmo más fuerte	239
Creando un hash de contraseña	240
Sal para el hash de contraseña	241
Verificando una contraseña contra un hash	241
Capítulo 49: Galletas.....	243
Introducción	243
Sintaxis	243
Parámetros	243
Observaciones	244
Examples	244
Configuración de una cookie	244
Recuperar una cookie	244
Modificar una cookie	245
Comprobando si una cookie está configurada	245
Eliminar una cookie	245
Capítulo 50: Generadores.....	247
Examples	247
¿Por qué usar un generador?	247

Reescribiendo números aleatorios () usando un generador.....	247
Leyendo un archivo grande con un generador.....	248
La palabra clave de rendimiento.....	248
Valores de rendimiento.....	249
Rendimiento de valores con claves.....	249
Uso de la función send () para pasar valores a un generador.....	250
Capítulo 51: Gerente de dependencia del compositor.....	252
Introducción.....	252
Sintaxis.....	252
Parámetros.....	252
Observaciones.....	252
Enlaces Útiles.....	252
Pocas sugerencias.....	253
Examples.....	253
¿Qué es el compositor?.....	253
Autocarga con compositor.....	254
Beneficios de usar Composer.....	255
Diferencia entre "instalación del compositor" y "actualización del compositor".....	255
composer update.....	255
composer install.....	256
Cuándo instalar y cuándo actualizar.....	256
Composer de comandos disponibles.....	256
Instalación.....	258
En la zona.....	258
Globalmente.....	258
Capítulo 52: Imaginario.....	260
Examples.....	260
Primeros pasos.....	260
Convertir imagen en base64 String.....	260
Capítulo 53: IMAP.....	262
Examples.....	262

Instalar extensión IMAP.....	262
Conectando a un buzón.....	262
Listar todas las carpetas en el buzón.....	264
Encontrar mensajes en el buzón.....	264
Capítulo 54: Instalación de un entorno PHP en Windows.....	267
Observaciones.....	267
Examples.....	267
Descargar e instalar XAMPP.....	267
¿Qué es XAMPP?.....	267
¿De dónde debería descargarlo?.....	267
¿Cómo instalar y dónde debo colocar mis archivos PHP / html?.....	267
Instale con el instalador provisto.....	268
Instalar desde el ZIP.....	268
Postinstalación.....	268
Manejo de archivos.....	268
Descarga, instala y usa WAMP.....	269
Instalar PHP y usarlo con IIS.....	270
Capítulo 55: Instalación en entornos Linux / Unix.....	272
Examples.....	272
Instalación de línea de comandos utilizando APT para PHP 7.....	272
Instalación en distribuciones Enterprise Linux (CentOS, Scientific Linux, etc.).....	272
Capítulo 56: Interfaz de línea de comandos (CLI).....	274
Examples.....	274
Manejo de argumentos.....	274
Manejo de entrada y salida.....	275
Códigos de retorno.....	276
Opciones de programa de manejo.....	276
Restrinja la ejecución del script a la línea de comando.....	278
Ejecutando tu guion.....	278
Diferencias de comportamiento en la línea de comando.....	279
Ejecutando servidor web incorporado.....	279
Edge Casos de getopt ().....	280

Capítulo 57: Inyección de dependencia	282
Introducción	282
Examples	282
Inyección Constructor	282
Inyección de Setter	283
Inyección de contenedores	284
Capítulo 58: Iteración de matriz	286
Sintaxis	286
Observaciones	286
Comparación de métodos para iterar una matriz	286
Examples	286
Iterando múltiples matrices juntas	286
Usando un índice incremental	287
Usando punteros de matriz interna	288
Usando each	288
Usando next	289
Usando foreach	289
Bucle directo	289
Lazo con llaves	289
Bucle por referencia	290
Concurrencia	290
Usando ArrayObject Iterator	291
Capítulo 59: JSON	292
Introducción	292
Sintaxis	292
Parámetros	292
Observaciones	292
Examples	293
Decodificando una cadena JSON	293
Codificar una cadena JSON	296
Argumentos	296

JSON_FORCE_OBJECT.....	297
JSON_HEX_TAG , JSON_HEX_AMP , JSON_HEX_APOS , JSON_HEX_QUOT.....	297
JSON_NUMERIC_CHECK.....	297
JSON_PRETTY_PRINT.....	298
JSON_UNESCAPED_SLASHES.....	298
JSON_UNESCAPED_UNICODE.....	298
JSON_PARTIAL_OUTPUT_ON_ERROR.....	299
JSON_PRESERVE_ZERO_FRACTION.....	299
JSON_UNESCAPED_LINE_TERMINATORS.....	299
Depuración de errores JSON.....	299
json_last_error_msg.....	300
json_last_error.....	301
Usando JsonSerializable en un objeto.....	301
Ejemplo de valores de propiedades.....	302
Usando propiedades privadas y protegidas con json_encode().....	302
Salida:.....	303
Encabezado json y la respuesta devuelta.....	303
Capítulo 60: Localización.....	305
Sintaxis.....	305
Examples.....	305
Localizando cadenas con gettext ().....	305
Capítulo 61: Los operadores.....	307
Introducción.....	307
Observaciones.....	307
Examples.....	308
Operadores de cadena (. Y .=).....	308
Asignación básica (=).....	308
Asignación combinada (+ = etc).....	309
Modificación de la precedencia del operador (entre paréntesis).....	310
Asociación.....	310
Asociación izquierda.....	310

Asociación correcta	310
Operadores de comparación	311
Igualdad	311
Comparacion de objetos	311
Otros operadores de uso común	311
Operador de la nave espacial (<=>)	313
Operador coalescente nulo (??)	313
instanceof (operador de tipo)	314
Advertencias	315
Versiones anteriores de PHP (antes de 5.0)	316
Operador Ternario (? :)	316
Incremento (++) y operadores decrecientes (-)	317
Operador de Ejecución (`)	317
Operadores lógicos (&& / AND y / OR)	318
Operadores de Bitwise	318
Prefijo de operadores bitwise	318
Operadores de máscara de bits	318
Ejemplos de usos de las máscaras de bits	319
Operadores de cambio de bits	320
Ejemplos de usos del desplazamiento de bits:	320
Operadores de objetos y clases	321
Capítulo 62: Los tipos	323
Examples	323
Enteros	323
Instrumentos de cuerda	324
Cita única	324
Doble citado	324
Heredoc	325
Ahoradoc	325
Booleano	325
Flotador	327

Advertencia	327
Callable	328
Nulo	328
Variable nula vs indefinida	329
Comparación de tipos	329
Tipo de fundición	330
Recursos	331
Tipo malabarismo	331
Capítulo 63: Manejo de archivos	332
Sintaxis	332
Parámetros	332
Observaciones	332
Sintaxis de nombre de archivo	332
Examples	333
Eliminar archivos y directorios	333
Borrando archivos	333
Eliminando directorios, con borrado recursivo	333
Funciones de conveniencia	334
Raw directo IO	334
CSV IO	334
Leyendo un archivo a stdout directamente	335
O desde un puntero de archivo	335
Leyendo un archivo en una matriz	335
Obteniendo información del archivo	336
Compruebe si una ruta es un directorio o un archivo	336
Comprobando el tipo de archivo	336
Comprobando legibilidad y capacidad de escritura	337
Comprobando el acceso a los archivos / modificar el tiempo	337
Obtener partes de ruta con fileinfo	337
Minimiza el uso de la memoria al tratar con archivos grandes	338

Archivo basado en flujo IO.....	339
Abriendo un arroyo.....	339
Leyendo.....	340
Líneas de lectura	341
Leyendo todo lo que queda.....	341
Ajuste de la posición del puntero del archivo.....	341
Escritura.....	341
Mover y copiar archivos y directorios.....	342
Copiando documentos.....	342
Copiando directorios, con recursion.....	342
Renombrando / Moviendo.....	342
Capítulo 64: Manejo de excepciones y reporte de errores.....	344
Examples.....	344
Configuración de informes de errores y dónde mostrarlos.....	344
Manejo de excepciones y errores.....	344
trata de atraparlo.....	345
La captura de diferentes tipos de excepción.....	345
finalmente.....	345
lanzable.....	346
Registro de errores fatales.....	346
Capítulo 65: Manipulación De Cabeceras.....	348
Examples.....	348
Configuración básica de un encabezado.....	348
Capítulo 66: Manipulando una matriz.....	350
Examples.....	350
Eliminar elementos de una matriz	350
Eliminando elementos terminales.....	350
Filtrando una matriz	351
Filtrado de valores no vacíos.....	351
Filtrado por devolución de llamada.....	351

Filtrado por índice	352
Índices en matriz filtrada	352
Añadiendo elemento al inicio del array.....	353
Lista blanca solo algunas claves de matriz	354
Ordenar una matriz	355
ordenar()	355
rsort ()	355
un tipo()	355
Arsort ()	356
ksort ()	356
krsort ()	356
natsort ()	357
natcasesort ()	357
barajar()	358
usort ()	358
uasort ()	359
uksort ()	359
Intercambiar valores con claves.....	360
Fusionar dos matrices en una matriz	360
Capítulo 67: Metodos magicos	362
Examples.....	362
__get (), __set (), __isset () y __unset ().....	362
Función vacía () y métodos mágicos.....	363
__construir () y __destruct ().....	363
__Encadenar().....	364
__invocar().....	364
__call () y __callStatic ().....	365
Ejemplo:.....	366
__sleep () y __wakeup ().....	367
__información de depuración().....	367
__clon().....	368

Capítulo 68: mongo-php.....	369
Sintaxis.....	369
Examples.....	369
Todo entre MongoDB y Php.....	369
Capítulo 69: Multiprocesamiento.....	372
Examples.....	372
Multiprocesamiento utilizando funciones de horquilla incorporadas.....	372
Creando proceso hijo usando tenedor.....	372
Comunicación entre procesos.....	373
Capítulo 70: Patrones de diseño.....	375
Introducción.....	375
Examples.....	375
Método de encadenamiento en PHP.....	375
Cuando usarlo.....	376
Notas adicionales.....	376
Separación de consulta de comando.....	376
Getters.....	376
Ley de Demeter e impacto en las pruebas.....	376
Capítulo 71: PHP incorporado en el servidor.....	378
Introducción.....	378
Parámetros.....	378
Observaciones.....	378
Examples.....	378
Ejecutando el servidor incorporado.....	378
Servidor incorporado con directorio específico y script de enrutador.....	379
Capítulo 72: PHP MySQLi.....	380
Introducción.....	380
Observaciones.....	380
Características.....	380
Alternativas.....	380
Examples.....	380

MySQLi Connect	380
Consulta de MySQLi	381
Recorrer los resultados de MySQLi	382
Conexión cercana	383
Declaraciones preparadas en MySQLi	383
Cuerdas de escape	384
MySQLi Insertar ID	385
Depuración de SQL en MySQLi	386
Cómo obtener datos de una declaración preparada	387
Declaraciones preparadas	387
Vinculación de resultados	387
¿Qué pasa si no puedo instalar mysqlnd ?	388
Capítulo 73: php mysqli filas afectadas devuelve 0 cuando debería devolver un entero positivo	390
Introducción	390
Examples	390
PHP \$ stmt->affected_rows devolviendo intermitentemente 0 cuando debería devolver un ente	390
Capítulo 74: PHPDoc	391
Sintaxis	391
Observaciones	391
Examples	392
Añadiendo metadatos a las funciones	392
Añadiendo metadatos a archivos	392
Heredar metadatos de estructuras padre	393
Describiendo una variable	393
Describiendo parámetros	394
Colecciones	395
Sintaxis de genéricos	395
Ejemplos	395
Capítulo 75: Primer de carga automática	397
Sintaxis	397
Observaciones	397
Examples	397

Definición de clase en línea, no requiere carga	397
Carga manual de clases con requerimiento.....	397
La carga automática reemplaza la carga de definición de clase manual.....	398
Autocarga como parte de una solución marco.....	398
Autocarga con compositor.....	399
Capítulo 76: Problemas de malabarismo de tipo y comparación no estricta	401
Examples.....	401
¿Qué es el tipo de malabarismo?.....	401
Leyendo de un archivo.....	402
Cambiar sorpresas.....	403
Casting explícito.....	403
Evitar el switch.....	403
Tipificación estricta	404
Capítulo 77: Procesamiento de imágenes con GD.....	405
Observaciones.....	405
Examples.....	405
Creando una imagen.....	405
Convertir una imagen.....	405
Salida de imagen.....	406
Guardando en un archivo.....	406
Salida como una respuesta HTTP.....	406
Escribir en una variable.....	406
Utilizando OB (buffer de salida).....	407
Usando envoltorios de flujo.....	407
Ejemplo de uso.....	408
Recorte de imagen y cambio de tamaño.....	408
Capítulo 78: Procesando múltiples matrices juntos.....	411
Examples.....	411
Fusionar o concatenar matrices.....	411
Intersección de matriz	411
Combinando dos matrices (claves de una, valores de otra).....	412
Cambio de una matriz multidimensional a matriz asociativa.....	412

Capítulo 79: Programación asíncrona	414
Examples	414
Ventajas de los generadores	414
Usando el bucle de evento Icicle	414
Usando el bucle de eventos Amp	415
Generando procesos no bloqueantes con proc_open ()	415
Lectura de puerto serie con evento y DIO	417
Pruebas	419
Cliente HTTP basado en la extensión del evento	419
http-client.php	419
prueba.php	421
Uso	421
Cliente HTTP basado en la extensión Ev	422
http-client.php	422
Pruebas	426
Capítulo 80: Programación Funcional	428
Introducción	428
Examples	428
Asignación a variables	428
Usando variables externas	428
Pasando una función de devolución de llamada como parámetro	429
Estilo procesal:	429
Estilo orientado a objetos:	429
Estilo orientado a objetos utilizando un método estático:	429
Usando funciones incorporadas como devoluciones de llamada	430
Función anónima	430
Alcance	431
Cierres	431
Funciones puras	433
Objetos como función	433
Métodos funcionales comunes en PHP	434
Cartografía	434

Reducción (o plegado)	434
Filtración	434
Capítulo 81: PSR	435
Introducción	435
Examples	435
PSR-4: Autoloader	435
PSR-1: Estándar de codificación básica	436
PSR-8: Interfaz Huggable	436
Capítulo 82: Publicación por entregas	438
Sintaxis	438
Parámetros	438
Observaciones	438
Examples	439
Serialización de diferentes tipos	439
Serializar una cuerda	439
Serializacion de un doble	439
Serializando un flotador	439
Serialización de un entero	439
Serializacion de un booleano	439
Serializacion nula	440
Serializando una matriz	440
Serialización de un objeto	440
Tenga en cuenta que los cierres no pueden ser serializados:	441
Problemas de seguridad con unserialize	441
Posibles ataques	441
Inyección de objetos PHP	441
Capítulo 83: Rasgos	444
Examples	444
Rasgos para facilitar la reutilización de código horizontal	444
La resolución de conflictos	445
Uso de múltiples rasgos	446

Modificación de la visibilidad del método.....	447
¿Qué es un rasgo?.....	447
¿Cuándo debo usar un rasgo?.....	448
Rasgos para mantener las clases limpias.....	449
Implementando un Singleton usando Rasgos.....	449
Capítulo 84: Recetas.....	452
Introducción.....	452
Examples.....	452
Crear un contador de visitas al sitio.....	452
Capítulo 85: Recopilación de errores y advertencias.....	453
Examples.....	453
Aviso: índice indefinido.....	453
Advertencia: no se puede modificar la información del encabezado - los encabezados ya envi.....	453
Error de análisis: error de sintaxis, T_PAAMAYIM_NEKUDOTAYIM inesperado.....	454
Capítulo 86: Referencias.....	455
Sintaxis.....	455
Observaciones.....	455
Examples.....	455
Asignar por referencia.....	455
Devolución por referencia.....	456
Notas.....	457
Pasar por referencia.....	457
Arrays.....	457
Funciones.....	458
Capítulo 87: Reflexión.....	460
Examples.....	460
Acceso a variables de miembros privados y protegidos.....	460
Detección de características de clases u objetos.....	462
Prueba de métodos privados / protegidos.....	463
Capítulo 88: Salida del valor de una variable.....	465
Introducción.....	465

Observaciones.....	465
Examples.....	465
hacer eco e imprimir.....	465
Notación abreviada de echo.....	466
Prioridad de print	466
Diferencias entre echo e print	467
Salida de una vista estructurada de arrays y objetos.....	467
print_r() - Arreglos y objetos de salida para la depuración.....	467
var_dump() : genera información de depuración legible para el ser humano sobre el contenido.....	468
var_export() - Código PHP salida válido.....	469
printf vs sprintf.....	469
Concatenación de cuerdas con eco.....	470
Concatenación de cadenas vs pasar múltiples argumentos a echo.....	471
Salida de enteros grandes.....	471
Genere una matriz multidimensional con índice y valor e imprima en la tabla.....	472
Capítulo 89: Seguridad.....	473
Introducción.....	473
Observaciones.....	473
Examples.....	473
Error al reportar.....	473
Una solución rápida.....	473
Errores de manejo.....	473
Secuencias de comandos entre sitios (XSS).....	474
Problema.....	474
Solución.....	475
Funciones de filtro.....	475
Codificación HTML.....	475
Codificación URL.....	475
Usando bibliotecas externas especializadas o listas de OWASP AntiSamy.....	476
Inclusión de archivos.....	476
Inclusión remota de archivos.....	476
Inclusión de archivos locales.....	476

Solución a RFI y LFI:	476
Inyección de línea de comando.....	477
Problema.....	477
Solución.....	477
Fuga de la versión de PHP.....	478
Etiquetas de desmontaje.....	478
Ejemplo básico.....	479
Permitir etiquetas.....	479
Aviso (s).....	479
Solicitud de falsificación entre sitios.....	479
Problema.....	479
Solución.....	480
Código de muestra.....	480
Subiendo archivos.....	481
Los datos subidos:.....	481
Explotando el nombre del archivo.....	481
Obtención segura del nombre y extensión del archivo.....	482
Validación de tipo mime.....	483
Lista blanca de tus subidas.....	483
Capítulo 90: Serialización de objetos.....	484
Sintaxis.....	484
Observaciones.....	484
Examples.....	484
Serializar / No serializar.....	484
La interfaz serializable.....	484
Capítulo 91: Servidor SOAP.....	486
Sintaxis.....	486
Examples.....	486
Servidor SOAP básico.....	486
Capítulo 92: Sesiones.....	487
Sintaxis.....	487

Observaciones.....	487
Examples.....	487
Manipulación de datos de sesión.....	487
Advertencia:.....	488
Destruye toda una sesión.....	488
Opciones de session_start ().....	489
Nombre de sesión.....	489
Comprobando si las cookies de sesión han sido creadas.....	489
Cambio de nombre de sesión.....	490
Bloqueo de sesión.....	490
Inicio de sesión segura sin errores.....	491
Capítulo 93: SimpleXML.....	492
Examples.....	492
Cargando datos XML en simplexml.....	492
Cargando desde la cuerda.....	492
Cargando desde archivo.....	492
Capítulo 94: Sintaxis alternativa para estructuras de control.....	493
Sintaxis.....	493
Observaciones.....	493
Examples.....	493
Alternativa para la declaración.....	493
Alternativa mientras que la declaración.....	493
Declaración foreach alternativa.....	494
Declaración de cambio alternativo.....	494
Alternativa si / else declaración.....	494
Capítulo 95: Soporte Unicode en PHP.....	496
Examples.....	496
Conversión de caracteres Unicode a formato "\ uxxxx" usando PHP.....	496
Cómo utilizar :.....	496
Salida :.....	496
Conversión de caracteres Unicode a su valor numérico y / o entidades HTML usando PHP.....	496

Cómo utilizar :	497
Salida :	498
Extensión internacional para soporte de Unicode.....	498
Capítulo 96: SQLite3.....	499
Examples.....	499
Consultar una base de datos.....	499
Recuperando un solo resultado.....	499
Tutorial de inicio rápido de SQLite3.....	499
Creación / apertura de una base de datos.....	499
Creando una mesa.....	500
Insertar datos de muestra.....	500
Recuperacion de datos.....	500
Shorthands.....	501
Limpiar.....	501
Capítulo 97: Tipo de insinuación.....	503
Sintaxis.....	503
Observaciones.....	503
Examples.....	503
Tipografía de tipos escalares, matrices y callables.....	503
Una excepción: tipos especiales.....	505
Tipo de sugerencia de objetos genéricos.....	505
Tipo de insinuación de clases e interfaces.....	506
Indicación de tipo de clase.....	506
Sugerencia de tipo de interfaz.....	507
Sugerencias de auto tipo.....	507
Tipo de sugerencia de no retorno (nulo).....	507
Consejos de tipo anulable.....	508
Parámetros.....	508
Valores de retorno.....	508
Capítulo 98: Trabajando con fechas y horarios.....	510

Sintaxis.....	510
Examples.....	510
Analizar las descripciones de fechas en inglés en un formato de fecha	510
Convertir una fecha en otro formato.....	510
Uso de constantes predefinidas para el formato de fecha	512
Consiguiendo la diferencia entre dos fechas / horarios.....	513
Capítulo 99: URLs.....	515
Examples.....	515
Analizar una URL.....	515
Redireccionando a otra URL	515
Construye una cadena de consulta codificada en URL desde una matriz.....	516
Capítulo 100: Usando cURL en PHP.....	518
Sintaxis.....	518
Parámetros.....	518
Examples.....	518
Uso básico (solicitudes GET).....	518
Solicitudes POST.....	519
Usando multi_curl para hacer múltiples solicitudes POST.....	519
Creando y enviando una solicitud con un método personalizado.....	521
Uso de cookies.....	521
Envío de datos multidimensionales y varios archivos con CurlFile en una sola solicitud.....	522
Obtén y establece encabezados http personalizados en php.....	525
Capítulo 101: Usando Redis con PHP.....	527
Examples.....	527
Instalando PHP Redis en Ubuntu.....	527
Conectando a una instancia de Redis.....	527
Ejecutando comandos redis en PHP.....	527
Capítulo 102: UTF-8.....	528
Observaciones.....	528
Examples.....	528
Entrada	528
Salida	528

Almacenamiento de datos y acceso.....	529
Capítulo 103: Utilizando MongoDB.....	531
Examples.....	531
Conectarse a MongoDB.....	531
Obtener un documento - findOne ().....	531
Obtener varios documentos - encontrar ().....	531
Insertar documento.....	531
Actualizar un documento.....	532
Borrar un documento.....	532
Capítulo 104: Utilizando SQLSRV.....	533
Observaciones.....	533
Examples.....	533
Creando una conexión.....	533
Haciendo una consulta simple.....	534
Invocando un procedimiento almacenado.....	534
Haciendo una consulta parametrizada.....	534
Obteniendo resultados de consultas.....	535
sqlsrv_fetch_array ().....	535
sqlsrv_fetch_object ().....	535
sqlsrv_fetch ().....	535
Recuperar mensajes de error.....	536
Capítulo 105: Variables.....	537
Sintaxis.....	537
Observaciones.....	537
Verificación de tipos.....	537
Examples.....	538
Acceso a una variable dinámicamente por nombre (variables variables).....	538
Diferencias entre PHP5 y PHP7.....	539
Caso 1: \$\$foo['bar']['baz'].....	540
Caso 2: \$foo->\$bar['baz'].....	540
Caso 3: \$foo->\$bar["baz"]().....	540
Caso 4: Foo::\$bar['baz']().....	540

Tipos de datos.....	540
Nulo.....	540
Booleano.....	541
Entero.....	541
Flotador.....	541
Formación.....	541
Cuerda.....	542
Objeto.....	542
Recurso.....	542
Mejores prácticas de variables globales.....	543
Obteniendo todas las variables definidas.....	545
Valores por defecto de variables no inicializadas.....	545
Valor de verdad variable y operador idéntico.....	546
Capítulo 106: Variables Superglobales PHP.....	549
Introducción.....	549
Examples.....	549
PHP5 SuperGlobals.....	549
Subglobales explicados.....	552
Introducción.....	552
¿Qué es un superglobal?.....	552
Cuéntame más cuéntame más.....	553
\$GLOBALS.....	553
Volverse global.....	554
\$_SERVER.....	554
\$_GET.....	556
\$_POST.....	556
\$_FILES.....	557
\$_COOKIE.....	559
\$_SESSION.....	560
\$_REQUEST.....	560
\$_ENV.....	560

Capítulo 107: Websockets.....	562
Introducción.....	562
Examples.....	562
Servidor TCP / IP simple.....	562
Capítulo 108: XML.....	564
Examples.....	564
Crea un archivo XML usando XMLWriter.....	564
Leer un documento XML con DOMDocument.....	564
Crea un XML utilizando DomDocument.....	565
Lee un documento XML con SimpleXML.....	567
Aprovechando XML con la biblioteca SimpleXML de PHP.....	568
Capítulo 109: YAML en PHP.....	571
Examples.....	571
Instalación de la extensión YAML.....	571
Usando YAML para almacenar la configuración de la aplicación.....	571
Creditos.....	573

Acerca de

You can share this PDF with anyone you feel could benefit from it, download the latest version from: [php](#)

It is an unofficial and free PHP ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PHP.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con PHP

Observaciones



PHP (acrónimo recursivo para PHP: preprocesador de hipertexto) es un lenguaje de programación de código abierto muy utilizado. Es especialmente adecuado para el desarrollo web. Lo único de PHP es que sirve tanto para principiantes como para desarrolladores experimentados. Tiene una barrera de entrada baja, por lo que es fácil comenzar y, al mismo tiempo, proporciona características avanzadas ofrecidas en otros lenguajes de programación.

Fuente abierta

Es un proyecto de código abierto. Siéntete libre de [involucrarte](#).

Especificación de idioma

PHP tiene una [especificación de lenguaje](#).

Versiones soportadas

Actualmente, hay tres [versiones soportadas](#): 5.6, 7.0 y 7.1.

Cada rama de lanzamiento de PHP es totalmente compatible durante dos años a partir de su lanzamiento estable inicial. Después de este período de dos años de soporte activo, cada sucursal recibe soporte por un año adicional solo para problemas de seguridad críticos. Las liberaciones durante este período se realizan según sea necesario: puede haber varias liberaciones puntuales, o ninguna, según el número de informes.

Versiones no soportadas

Una vez que se completan los tres años de soporte, la sucursal llega al final de su vida útil y ya no es compatible.

Una [tabla de ramas de fin de vida](#) está disponible.

Rastreador de problemas

Los errores y otros problemas se rastrean en <https://bugs.php.net/>.

Listas de correo

Las discusiones sobre el desarrollo y uso de PHP se llevan a cabo en las [listas de correo de PHP](#).

Documentación oficial

Por favor ayuda a mantener o traducir la [documentación oficial de PHP](#) .

Puede usar el editor en [edit.php.net](#) . Echa un vistazo a nuestra [guía para los colaboradores](#) .

Versiones

PHP 7.x

Versión	Apoyado hasta	Fecha de lanzamiento
7.1	2019-12-01	2016-12-01
7.0	2018-12-03	2015-12-03

PHP 5.x

Versión	Apoyado hasta	Fecha de lanzamiento
5.6	2018-12-31	2014-08-28
5.5	2016-07-21	2013-06-20
5.4	2015-09-03	2012-03-01
5.3	2014-08-14	30-06-2009
5.2	2011-01-06	2006-11-02
5.1	2006-08-24	2005-11-24
5.0	2005-09-05	2004-07-13

PHP 4.x

Versión	Apoyado hasta	Fecha de lanzamiento
4.4	2008-08-07	2005-07-11
4.3	2005-03-31	2002-12-27
4.2	2002-09-06	2002-04-22
4.1	2002-03-12	2001-12-10
4.0	2001-06-23	2000-05-22

Versiones heredadas

Versión	Apoyado hasta	Fecha de lanzamiento
3.0	2000-10-20	1998-06-06
2.0		1997-11-01
1.0		1995-06-08

Examples

Salida HTML del servidor web

PHP se puede utilizar para agregar contenido a archivos HTML. Si bien el HTML se procesa directamente mediante un navegador web, los scripts PHP se ejecutan en un servidor web y el HTML resultante se envía al navegador.

El siguiente código HTML contiene una declaración de PHP que agregará `Hello World!` a la salida:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP!</title>
  </head>
  <body>
    <p><?php echo "Hello world!"; ?></p>
  </body>
</html>
```

Cuando se guarde como un script PHP y lo ejecute un servidor web, se enviará el siguiente HTML al navegador del usuario:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP!</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

PHP 5.x 5.4

`echo` también tiene una sintaxis de acceso directo, que le permite imprimir inmediatamente un valor. Antes de PHP 5.4.0, esta sintaxis corta solo funciona con la configuración de configuración `short_open_tag` habilitada.

Por ejemplo, considere el siguiente código:

```
<p><?= "Hello world!" ?></p>
```

Su salida es idéntica a la salida de lo siguiente:

```
<p><?php echo "Hello world!"; ?></p>
```

En las aplicaciones del mundo real, todos los datos de salida de PHP a una página HTML deben escaparse adecuadamente para evitar ataques de XSS ([secuencias de comandos entre sitios](#)) o daños en el texto.

Vea también: [Cadenas y PSR-1](#), que describe las mejores prácticas, incluido el uso adecuado de etiquetas cortas (`<?= ... ?>`).

Salida no HTML desde servidor web

En algunos casos, cuando se trabaja con un servidor web, puede ser necesario reemplazar el tipo de contenido predeterminado del servidor web. Puede haber casos en los que necesite enviar datos como `plain text`, `JSON` o `XML`, por ejemplo.

La función `header()` puede enviar un encabezado HTTP en bruto. Puede agregar el encabezado `Content-Type` para notificar al navegador el contenido que estamos enviando.

Considere el siguiente código, donde configuramos `Content-Type` como `text/plain`:

```
header("Content-Type: text/plain");
echo "Hello World";
```

Esto producirá un documento de texto plano con el siguiente contenido:

Hola Mundo

Para producir contenido `JSON`, use el tipo de contenido `application/json` lugar:

```
header("Content-Type: application/json");

// Create a PHP data array.
$data = ["response" => "Hello World"];

// json_encode will convert it to a valid JSON string.
echo json_encode($data);
```

Esto producirá un documento de tipo `application/json` con el siguiente contenido:

```
{"respuesta": "Hola mundo"}
```

Tenga en cuenta que se debe llamar a la función `header()` antes de que PHP produzca cualquier salida, o el servidor web ya habrá enviado encabezados para la respuesta. Por lo tanto, considere el siguiente código:

```
// Error: We cannot send any output before the headers
```

```
echo "Hello";  
  
// All headers must be sent before ANY PHP output  
header("Content-Type: text/plain");  
echo "World";
```

Esto producirá una advertencia:

Advertencia: no se puede modificar la información del encabezado: los encabezados ya enviados por (la salida comenzó en /dir/example.php:2) en **/dir/example.php** en la línea 3

Cuando se utiliza `header()`, su salida debe ser el primer byte que se envía desde el servidor. Por esta razón, es importante no tener líneas o espacios vacíos al principio del archivo antes de la etiqueta de apertura de PHP `<?php`. Por la misma razón, se considera la mejor práctica (ver [PSR-2](#)) omitir la etiqueta de cierre de PHP `?>` De los archivos que contienen solo PHP y de los bloques de código PHP al final de un archivo.

Vea la [sección de almacenamiento en búfer de salida](#) para aprender cómo "capturar" su contenido en una variable para generar más tarde, por ejemplo, después de generar encabezados.

¡Hola Mundo!

La construcción de lenguaje más utilizada para imprimir la salida en PHP es `echo`:

```
echo "Hello, World!\n";
```

Alternativamente, también puede utilizar `print`:

```
print "Hello, World!\n";
```

Ambas declaraciones realizan la misma función, con pequeñas diferencias:

- `echo` tiene un retorno `void`, mientras que `print` devuelve un `int` con un valor de 1
- `echo` puede tomar múltiples argumentos (sin paréntesis solamente), mientras que `print` solo toma un argumento
- `echo` es un [poco más rápido](#) que la `print`

Tanto el `echo` como el `print` son construcciones de lenguaje, no funciones. Eso significa que no requieren paréntesis alrededor de sus argumentos. Para consistencia estética con funciones, se pueden incluir paréntesis. Los ejemplos extensos del uso del `echo` y la `print` están [disponibles en otros lugares](#).

El `printf` estilo C y las funciones relacionadas también están disponibles, como en el siguiente ejemplo:

```
printf("%s\n", "Hello, World!");
```

Consulte [Generar el valor de una variable](#) para obtener una introducción completa de las variables de salida en PHP.

Separación de instrucciones

Al igual que la mayoría de los otros lenguajes de estilo C, cada declaración termina con un punto y coma. Además, se usa una etiqueta de cierre para terminar la última línea de código del bloque PHP.

Si la última línea de código PHP termina con un punto y coma, la etiqueta de cierre es opcional si no hay ningún código después de esa línea final de código. Por ejemplo, podemos dejar la etiqueta de cierre después del `echo "No error";` en el siguiente ejemplo:

```
<?php echo "No error"; // no closing tag is needed as long as there is no code below
```

Sin embargo, si hay algún otro código que sigue a su bloque de código PHP, la etiqueta de cierre ya no es opcional:

```
<?php echo "This will cause an error if you leave out the closing tag"; ?>
<html>
  <body>
    </body>
</html>
```

También podemos omitir el punto y coma de la última instrucción en un bloque de código PHP si ese bloque de código tiene una etiqueta de cierre:

```
<?php echo "I hope this helps! :D";
echo "No error" ?>
```

En general, se recomienda usar siempre un punto y coma y una etiqueta de cierre para cada bloque de código PHP, excepto el último bloque de código PHP, si no hay más código que siga ese bloque de código PHP.

Por lo tanto, su código básicamente debería verse así:

```
<?php
  echo "Here we use a semicolon!";
  echo "Here as well!";
  echo "Here as well!";
  echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
  echo "Here we use a semicolon!";
  echo "Here as well!";
  echo "Here as well!";
  echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
  echo "Here we use a semicolon!";
```

```
echo "Here as well!";
echo "Here as well!";
echo "Here we use a semicolon but leave out the closing tag";
```

PHP CLI

PHP también se puede ejecutar desde la línea de comandos directamente mediante la CLI (interfaz de línea de comandos).

CLI es básicamente lo mismo que PHP desde los servidores web, excepto algunas diferencias en términos de entrada y salida estándar.

Disparando

La CLI de PHP permite cuatro formas de ejecutar código PHP:

1. Entrada estándar. Ejecute el comando `php` sin ningún argumento, pero incluya código PHP en él:

```
echo '<?php echo "Hello world!";' | php
```

2. Nombre de archivo como argumento. Ejecute el comando `php` con el nombre de un archivo fuente PHP como primer argumento:

```
php hello_world.php
```

3. Código como argumento. Use la opción `-r` en el comando `php`, seguido del código para ejecutar. No se requieren las etiquetas `<?php open`, ya que todo en el argumento se considera como código PHP:

```
php -r 'echo "Hello world!";'
```

4. Shell interactivo Use la opción `-a` en el comando `php` para iniciar un shell interactivo. Luego, escribe (o pega) el código PHP y pulsa `devolver`:

```
$ php -a
Interactive mode enabled
php > echo "Hello world!";
Hello world!
```

Salida

Todas las funciones o controles que producen resultados HTML en el servidor web PHP se pueden usar para producir resultados en la secuencia `stdout` (descriptor de archivo 1), y todas las acciones que producen resultados en los registros de errores en el servidor web PHP producirán resultados en la secuencia `stderr` (archivo descriptor 2).

```
<?php
echo "Stdout 1\n";
trigger_error("Stderr 2\n");
print_r("Stdout 3\n");
fwrite(STDERR, "Stderr 4\n");
throw new RuntimeException("Stderr 5\n");
?>
Stdout 6
```

Línea de comandos de shell

```
$ php Example.php 2>stderr.log >stdout.log; \
> echo STDOUT; cat stdout.log; echo; \
> echo STDERR; cat stderr.log\

STDOUT
Stdout 1
Stdout 3

STDERR
Stderr 4
PHP Notice:  Stderr 2
in /Example.php on line 3
PHP Fatal error:  Uncaught RuntimeException: Stderr 5
in /Example.php:6
Stack trace:
#0 {main}
    thrown in /Example.php on line 6
```

Entrada

Ver: [Interfaz de línea de comandos \(CLI\)](#)

Servidor incorporado de PHP

PHP 5.4+ viene con un servidor de desarrollo incorporado. Se puede usar para ejecutar aplicaciones sin tener que instalar un servidor HTTP de producción como nginx o Apache. El servidor incorporado solo está diseñado para ser utilizado con fines de desarrollo y prueba.

Se puede iniciar utilizando la bandera `-S` :

```
php -S <host/ip>:<port>
```

Ejemplo de uso

1. Crea un archivo `index.php` que contenga:

```
<?php
echo "Hello World from built-in PHP server";
```

- Ejecute el comando `php -S localhost:8080` desde la línea de comandos. No incluya `http://`. Esto iniciará la escucha del servidor web en el puerto 8080 utilizando el directorio actual en el que se encuentra como raíz del documento.
- Abra el navegador y navegue a `http://localhost:8080`. Deberías ver tu página de "Hola Mundo".

Configuración

Para anular la raíz de documento predeterminada (es decir, el directorio actual), use la `-t` :

```
php -S <host/ip>:<port> -t <directory>
```

Por ejemplo, si tiene un directorio `public/` en su proyecto, puede servir su proyecto desde ese directorio utilizando `php -S localhost:8080 -t public/`.

Troncos

Cada vez que se realiza una solicitud desde el servidor de desarrollo, una entrada de registro como la que se muestra a continuación se escribe en la línea de comandos.

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Etiquetas PHP

Hay tres tipos de etiquetas para denotar bloques de PHP en un archivo. El analizador de PHP está buscando las etiquetas de apertura y (si están presentes) para delimitar el código a interpretar.

Etiquetas estándar

Estas etiquetas son el método estándar para incrustar código PHP en un archivo.

```
<?php  
    echo "Hello World";  
?>
```

PHP 5.x 5.4

Etiquetas de eco

Estas etiquetas están disponibles en todas las versiones de PHP, y desde PHP 5.4 siempre están habilitadas. En versiones anteriores, las etiquetas de eco solo podían activarse junto con

etiquetas cortas.

```
<?= "Hello World" ?>
```

Etiquetas cortas

Puede deshabilitar o habilitar estas etiquetas con la opción `short_open_tag`.

```
<?
    echo "Hello World";
?>
```

Etiquetas cortas:

- no están permitidos en todos los principales [estándares de codificación PHP](#)
- Se desalientan en [la documentación oficial](#).
- están deshabilitados por defecto en la mayoría de las distribuciones
- interferir con las instrucciones de procesamiento de XML en línea
- No se aceptan en las presentaciones de código por la mayoría de los proyectos de código abierto.

PHP 5.x 5.6

Etiquetas ASP

Al habilitar la opción `asp_tags`, se pueden usar etiquetas de estilo ASP.

```
<%
    echo "Hello World";
%>
```

Estas son una peculiaridad histórica y nunca deben usarse. Fueron eliminados en PHP 7.0.

Lea Empezando con PHP en línea: <https://riptutorial.com/es/php/topic/189/empezando-con-php>

Capítulo 2: Actuación

Examples

Perfilando con XHProf

XHProf es un generador de perfiles PHP originalmente escrito por Facebook, para proporcionar una alternativa más liviana a XDebug.

Después de instalar el módulo PHP `xhprof`, la creación de perfiles se puede habilitar / deshabilitar desde el código PHP:

```
xhprof_enable();
doSlowOperation();
$profile_data = xhprof_disable();
```

La matriz devuelta contendrá datos sobre el número de llamadas, el tiempo de CPU y el uso de memoria de cada función a la que se ha accedido dentro de `doSlowOperation()`.

`xhprof_sample_enable()` / `xhprof_sample_disable()` puede usarse como una opción más liviana que solo registrará la información de perfiles para una fracción de las solicitudes (y en un formato diferente).

XHProf tiene algunas funciones de ayuda (en su mayoría no documentadas) para mostrar los datos ([ver ejemplo](#)), o puede usar otras herramientas para visualizarlos (el blog platform.sh [tiene un ejemplo](#)).

Uso de memoria

El límite de memoria en tiempo de ejecución de PHP se establece a través de la directiva INI `memory_limit`. Esta configuración evita que cualquier ejecución individual de PHP consuma demasiada memoria, agotándola para otros scripts y software del sistema. El límite de memoria predeterminado es de 128M y se puede cambiar en el archivo `php.ini` o en tiempo de ejecución. Puede configurarse para que no tenga límite, pero esto generalmente se considera una mala práctica.

El uso de memoria exacto utilizado durante el tiempo de ejecución se puede determinar llamando a `memory_get_usage()`. Devuelve el número de bytes de memoria asignados al script actualmente en ejecución. A partir de PHP 5.2, tiene un parámetro booleano opcional para obtener la memoria total del sistema asignada, a diferencia de la memoria que PHP está utilizando activamente.

```
<?php
echo memory_get_usage() . "\n";
// Outputs 350688 (or similar, depending on system and PHP version)

// Let's use up some RAM
$array = array_fill(0, 1000, 'abc');
```

```

echo memory_get_usage() . "\n";
// Outputs 387704

// Remove the array from memory
unset($array);

echo memory_get_usage() . "\n";
// Outputs 350784

```

Ahora `memory_get_usage` le da uso de memoria en el momento en que se ejecuta. Entre las llamadas a esta función, puede asignar y desasignar otras cosas en la memoria. Para obtener la cantidad máxima de memoria utilizada hasta cierto punto, llame a `memory_get_peak_usage()`.

```

<?php
echo memory_get_peak_usage() . "\n";
// 385688
$array = array_fill(0, 1000, 'abc');
echo memory_get_peak_usage() . "\n";
// 422736
unset($array);
echo memory_get_peak_usage() . "\n";
// 422776

```

Note que el valor solo subirá o se mantendrá constante.

Perfilando con Xdebug

Una extensión de PHP llamada Xdebug está disponible para ayudar en la [creación de perfiles de aplicaciones PHP](#), así como para la depuración en tiempo de ejecución. Cuando se ejecuta el generador de perfiles, la salida se escribe en un archivo en un formato binario llamado "cachegrind". Las aplicaciones están disponibles en cada plataforma para analizar estos archivos.

Para habilitar la creación de perfiles, instale la extensión y ajuste la configuración de `php.ini`. En nuestro ejemplo, ejecutaremos el perfil opcionalmente en función de un parámetro de solicitud. Esto nos permite mantener la configuración estática y activar el generador de perfiles solo cuando sea necesario.

```

// Set to 1 to turn it on for every request
xdebug.profiler_enable = 0
// Let's use a GET/POST parameter to turn on the profiler
xdebug.profiler_enable_trigger = 1
// The GET/POST value we will pass; empty for any value
xdebug.profiler_enable_trigger_value = ""
// Output cachegrind files to /tmp so our system cleans them up later
xdebug.profiler_output_dir = "/tmp"
xdebug.profiler_output_name = "cachegrind.out.%p"

```

A continuación, use un cliente web para realizar una solicitud a la URL de su aplicación que desea perfilar, por ejemplo,

```
http://example.com/article/1?XDEBUG_PROFILE=1
```

A medida que la página se procesa, se escribirá en un archivo con un nombre similar a

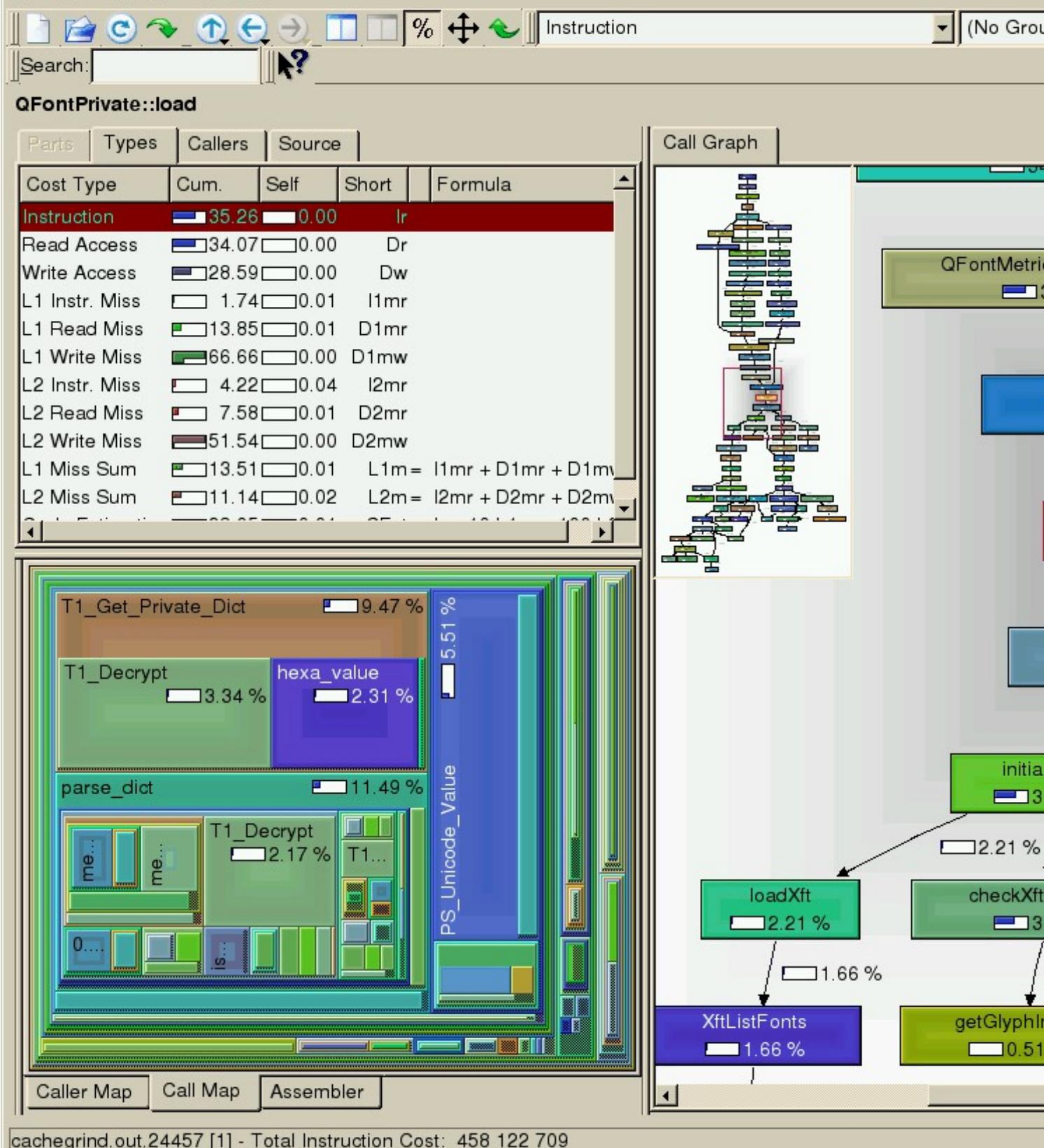
```
/tmp/cachegrind.out.12345
```

Tenga en cuenta que escribirá un archivo para cada solicitud / proceso de PHP que se ejecute. Por ejemplo, si desea analizar una publicación de formulario, se escribirá un perfil para la solicitud GET para mostrar el formulario HTML. El parámetro XDEBUG_PROFILE deberá pasar a la solicitud POST posterior para analizar la segunda solicitud que procesa el formulario. Por lo tanto, cuando se perfila, a veces es más fácil ejecutar curl para POSTAR un formulario directamente.

Una vez escrito, el caché de perfil puede ser leído por una aplicación como KCachegrind.

./cachegrind.out.24457 [kcachegrind] - KCachegrind

File View Go Settings Help



Esto mostrará información que incluye:

- Funciones ejecutadas
- Tiempo de llamada, tanto en sí mismo como inclusivo de llamadas de función posteriores
- Número de veces que se llama a cada función

- Gráficos de llamadas
- Enlaces al código fuente

Obviamente, el ajuste del rendimiento es muy específico para los casos de uso de cada aplicación. En general es bueno buscar:

- Llamadas repetidas a la misma función que no esperaría ver. Para las funciones que procesan y consultan datos, estas podrían ser las principales oportunidades para que su aplicación se almacene en caché.
- Funciones de ejecución lenta. ¿Dónde está la aplicación pasando la mayor parte de su tiempo? La mejor recompensa en la optimización del rendimiento es centrarse en aquellas partes de la aplicación que consumen más tiempo.

Nota : Xdebug, y en particular sus características de creación de perfiles, son muy intensivos en recursos y ralentizan la ejecución de PHP. Se recomienda no ejecutar estos en un entorno de servidor de producción.

Lea Actuación en línea: <https://riptutorial.com/es/php/topic/3723/actuacion>

Capítulo 3: Alcance variable

Introducción

El alcance variable se refiere a las regiones de código donde se puede acceder a una variable. Esto también se conoce como *visibilidad*. En PHP, los bloques de alcance se definen por funciones, clases y un alcance global disponible a través de una aplicación.

Examples

Variables globales definidas por el usuario

El alcance fuera de cualquier función o clase es el alcance global. Cuando un script PHP incluye otro (usando `include` o `require`), el alcance sigue siendo el mismo. Si un script se incluye fuera de cualquier función o clase, sus variables globales se incluyen en el mismo ámbito global, pero si un script se incluye dentro de una función, las variables en el script incluido están en el alcance de la función.

Dentro del alcance de una función o método de clase, la palabra clave `global` se puede usar para crear un acceso a variables globales definidas por el usuario.

```
<?php

$amount_of_log_calls = 0;

function log_message($message) {
    // Accessing global variable from function scope
    // requires this explicit statement
    global $amount_of_log_calls;

    // This change to the global variable is permanent
    $amount_of_log_calls += 1;

    echo $message;
}

// When in the global scope, regular global variables can be used
// without explicitly stating 'global $variable;'
echo $amount_of_log_calls; // 0

log_message("First log message!");
echo $amount_of_log_calls; // 1

log_message("Second log message!");
echo $amount_of_log_calls; // 2
```

Una segunda forma de acceder a las variables desde el ámbito global es usar la matriz `$GLOBALS` especial definida por PHP.

La matriz `$GLOBALS` es una matriz asociativa con el nombre de la variable global que es la

clave y el contenido de esa variable es el valor del elemento de la matriz. Observe cómo \$GLOBALS existe en cualquier ámbito, esto se debe a que \$GLOBALS es un superglobal.

Esto significa que la función `log_message()` podría reescribirse como:

```
function log_message($message) {
    // Access the global $amount_of_log_calls variable via the
    // $GLOBALS array. No need for 'global $GLOBALS;', since it
    // is a superglobal variable.
    $GLOBALS['amount_of_log_calls'] += 1;

    echo $messsage;
}
```

Uno podría preguntarse, ¿por qué usar la matriz \$GLOBALS cuando la palabra clave `global` también se puede usar para obtener el valor de una variable global? La razón principal es que el uso de la palabra clave `global` traerá la variable al alcance. Entonces no puede reutilizar el mismo nombre de variable en el ámbito local.

Variables superglobales

Las variables superglobales están definidas por PHP y siempre se pueden usar desde cualquier lugar sin la palabra clave `global`.

```
<?php

function getPostValue($key, $default = NULL) {
    // $_POST is a superglobal and can be used without
    // having to specify 'global $_POST;';
    if (isset($_POST[$key])) {
        return $_POST[$key];
    }

    return $default;
}

// retrieves $_POST['username']
echo getPostValue('username');

// retrieves $_POST['email'] and defaults to empty string
echo getPostValue('email', '');
```

Propiedades estáticas y variables

Las propiedades de clase estáticas que se definen con la visibilidad `public` son funcionalmente las mismas que las variables globales. Se puede acceder desde cualquier lugar donde se defina la clase.

```
class SomeClass {
    public static int $counter = 0;
}

// The static $counter variable can be read/written from anywhere
```

```
// and doesn't require an instantiation of the class
SomeClass::$counter += 1;
```

Las funciones también pueden definir variables estáticas dentro de su propio alcance. Estas variables estáticas persisten a través de múltiples llamadas a funciones, a diferencia de las variables regulares definidas en el alcance de una función. Esta puede ser una forma muy fácil y sencilla de implementar el patrón de diseño de Singleton:

```
class Singleton {
    public static function getInstance() {
        // Static variable $instance is not deleted when the function ends
        static $instance;

        // Second call to this function will not get into the if-statement,
        // Because an instance of Singleton is now stored in the $instance
        // variable and is persisted through multiple calls
        if (!$instance) {
            // First call to this function will reach this line,
            // because the $instance has only been declared, not initialized
            $instance = new Singleton();
        }

        return $instance;
    }
}

$instance1 = Singleton::getInstance();
$instance2 = Singleton::getInstance();

// Comparing objects with the '===' operator checks whether they are
// the same instance. Will print 'true', because the static $instance
// variable in the getInstance() method is persisted through multiple calls
var_dump($instance1 === $instance2);
```

Lea Alcance variable en línea: <https://riptutorial.com/es/php/topic/3426/alcance-variable>

Capítulo 4: Análisis de cuerdas

Observaciones

Se debe usar Regex para otros usos además de sacar cuerdas de las cuerdas o cortar las cuerdas de otra manera.

Examples

Dividir una cadena por separadores

`explode` y `strstr` son métodos más simples para obtener subcadenas por separadores.

Una cadena que contiene varias partes del texto que están separadas por un carácter común se puede dividir en partes con la función de `explode`.

```
$fruits = "apple,pear,grapefruit,cherry";
print_r(explode(",",$fruits)); // ['apple', 'pear', 'grapefruit', 'cherry']
```

El método también admite un parámetro de límite que se puede utilizar de la siguiente manera:

```
$fruits= 'apple,pear,grapefruit,cherry';
```

Si el parámetro límite es cero, esto se trata como 1.

```
print_r(explode(',',$fruits,0)); // ['apple,pear,grapefruit,cherry']
```

Si se establece límite y positivo, la matriz devuelta contendrá un máximo de elementos límite con el último elemento que contiene el resto de la cadena.

```
print_r(explode(',',$fruits,2)); // ['apple', 'pear,grapefruit,cherry']
```

Si el parámetro límite es negativo, se devuelven todos los componentes excepto el último límite.

```
print_r(explode(',',$fruits,-1)); // ['apple', 'pear', 'grapefruit']
```

`explode` se puede combinar con `list` para analizar una cadena en variables en una línea:

```
$email = "user@example.com";
list($name, $domain) = explode("@", $email);
```

Sin embargo, asegúrese de que el resultado de la `explode` contenga suficientes elementos, o se activará una advertencia de índice no definido.

`strstr` quita o solo devuelve la subcadena antes de la primera aparición de la aguja dada.

```
$string = "1:23:456";
echo json_encode(explode(":", $string)); // ["1", "23", "456"]
var_dump(strstr($string, ":")); // string(7) ":23:456"
var_dump(strstr($string, ":", true)); // string(1) "1"
```

Buscando una subcadena con strpos

`strpos` puede entenderse como el número de bytes en el pajar antes de la primera aparición de la aguja.

```
var_dump(strpos("haystack", "hay")); // int(0)
var_dump(strpos("haystack", "stack")); // int(3)
var_dump(strpos("haystack", "stackoverflow")); // bool(false)
```

Comprobando si existe una subcadena

Tenga cuidado con la comprobación contra VERDADERO o FALSO porque si se devuelve un índice de 0, una instrucción `if` lo verá como FALSO.

```
$pos = strpos("abcd", "a"); // $pos = 0;
$pos2 = strpos("abcd", "e"); // $pos2 = FALSE;

// Bad example of checking if a needle is found.
if($pos) { // 0 does not match with TRUE.
    echo "1. I found your string\n";
}
else {
    echo "1. I did not find your string\n";
}

// Working example of checking if needle is found.
if($pos !== FALSE) {
    echo "2. I found your string\n";
}
else {
    echo "2. I did not find your string\n";
}

// Checking if a needle is not found
if($pos2 === FALSE) {
    echo "3. I did not find your string\n";
}
else {
    echo "3. I found your string\n";
}
```

Salida de todo el ejemplo:

```
1. I did not find your string
2. I found your string
3. I did not find your string
```

Búsqueda a partir de un offset

```
// With offset we can search ignoring anything before the offset
$needle = "Hello";
$haystack = "Hello world! Hello World";

$pos = strpos($haystack, $needle, 1); // $pos = 13, not 0
```

Consigue todas las apariciones de una subcadena.

```
$haystack = "a baby, a cat, a donkey, a fish";
$needle = "a ";
$offsets = [];
// start searching from the beginning of the string
for($offset = 0;
    // If our offset is beyond the range of the
    // string, don't search anymore.
    // If this condition is not set, a warning will
    // be triggered if $haystack ends with $needle
    // and $needle is only one byte long.
    $offset < strlen($haystack); ){
    $pos = strpos($haystack, $needle, $offset);
    // we don't have anymore substrings
    if($pos === false) break;
    $offsets[] = $pos;
    // You may want to add strlen($needle) instead,
    // depending on whether you want to count "aaa"
    // as 1 or 2 "aa"s.
    $offset = $pos + 1;
}
echo json_encode($offsets); // [0,8,15,25]
```

Analizando la cadena usando expresiones regulares

`preg_match` puede usarse para analizar cadenas usando expresiones regulares. Las partes de la expresión entre paréntesis se denominan subpatrones y con ellas puede seleccionar partes individuales de la cadena.

```
$str = "<a href=\"http://example.org\">My Link</a>";
$pattern = "/<a href=\"(.*)\">(.*)</a>/";
$result = preg_match($pattern, $str, $matches);
if($result === 1) {
    // The string matches the expression
    print_r($matches);
} else if($result === 0) {
    // No match
} else {
    // Error occurred
}
```

Salida

```
Array
(
    [0] => <a href="http://example.org">My Link</a>
    [1] => http://example.org
    [2] => My Link
)
```

Subcadena

La subcadena devuelve la porción de cadena especificada por los parámetros de inicio y longitud.

```
var_dump(substr("Boo", 1)); // string(2) "oo"
```

Si existe la posibilidad de cumplir con cadenas de caracteres de múltiples bytes, sería más seguro usar `mb_substr`.

```
$cake = "cakeæøå";
var_dump(substr($cake, 0, 5)); // string(5) "cake "
var_dump(mb_substr($cake, 0, 5, 'UTF-8')); // string(6) "cakeæ"
```

Otra variante es la función `substr_replace`, que reemplaza el texto dentro de una parte de una cadena.

```
var_dump(substr_replace("Boo", "0", 1, 1)); // string(3) "B0o"
var_dump(substr_replace("Boo", "ts", strlen("Boo"))); // string(5) "Boots"
```

Digamos que quiere encontrar una palabra específica en una cadena y no quiere usar Regex.

```
$hi = "Hello World!";
$bye = "Goodbye cruel World!";

var_dump(strpos($hi, " ")); // int(5)
var_dump(strpos($bye, " ")); // int(7)

var_dump(substr($hi, 0, strpos($hi, " "))); // string(5) "Hello"
var_dump(substr($bye, -1 * (strlen($bye) - strpos($bye, " ")))); // string(13) " cruel World!"

// If the casing in the text is not important, then using strtolower helps to compare strings
var_dump(substr($hi, 0, strpos($hi, " ")) == 'hello'); // bool(false)
var_dump(strtolower(substr($hi, 0, strpos($hi, " "))) == 'hello'); // bool(true)
```

Otra opción es un análisis muy básico de un correo electrónico.

```
$email = "test@example.com";
$wrong = "foobar.co.uk";
$notld = "foo@bar";

$at = strpos($email, "@"); // int(4)
$wat = strpos($wrong, "@"); // bool(false)
$nat = strpos($notld, "@"); // int(3)
```

```

$domain = substr($email, $at + 1); // string(11) "example.com"
$womain = substr($wrong, $wat + 1); // string(11) "oobar.co.uk"
$nomain = substr($notld, $nat + 1); // string(3) "bar"

$dot = strpos($domain, "."); // int(7)
$wot = strpos($womain, "."); // int(5)
$not = strpos($nomain, "."); // bool(false)

$tld = substr($domain, $dot + 1); // string(3) "com"
$wld = substr($womain, $wot + 1); // string(5) "co.uk"
$nld = substr($nomain, $not + 1); // string(2) "ar"

// string(25) "test@example.com is valid"
if ($at && $dot) var_dump("$email is valid");
else var_dump("$email is invalid");

// string(21) "foobar.com is invalid"
if ($wat && $wot) var_dump("$wrong is valid");
else var_dump("$wrong is invalid");

// string(18) "foo@bar is invalid"
if ($nat && $not) var_dump("$notld is valid");
else var_dump("$notld is invalid");

// string(27) "foobar.co.uk is an UK email"
if ($tld == "co.uk") var_dump("$email is a UK address");
if ($wld == "co.uk") var_dump("$wrong is a UK address");
if ($nld == "co.uk") var_dump("$notld is a UK address");

```

O incluso poner "Continuar leyendo" o "..." al final de una propaganda

```

$blurb = "Lorem ipsum dolor sit amet";
$limit = 20;

var_dump(substr($blurb, 0, $limit - 3) . '...'); // string(20) "Lorem ipsum dolor..."

```

Lea Análisis de cuerdas en línea: <https://riptutorial.com/es/php/topic/2206/analisis-de-cuerdas>

Capítulo 5: Análisis de HTML

Examples

Analizar HTML desde una cadena

PHP implementa un analizador compatible con [DOM Nivel 2](#), lo que le permite trabajar con HTML utilizando métodos conocidos como `getElementById()` o `appendChild()`.

```
$html = '<html><body><span id="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
libxml_use_internal_errors(true);
$doc->loadHTML($html);

echo $doc->getElementById("text")->textContent;
```

Salidas:

```
Hello, World!
```

Tenga en cuenta que PHP emitirá advertencias sobre cualquier problema con el HTML, especialmente si está importando un fragmento de documento. Para evitar estas advertencias, indique a la biblioteca DOM (libxml) que maneje sus propios errores llamando a `libxml_use_internal_errors()` antes de importar su HTML. Luego puede usar `libxml_get_errors()` para manejar los errores si es necesario.

Utilizando XPath

```
$html = '<html><body><span class="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
$doc->loadHTML($html);

$xpath = new DOMXPath($doc);
$span = $xpath->query("//span[@class='text']")->item(0);

echo $span->textContent;
```

Salidas:

```
Hello, World!
```

SimpleXML

Presentación

- SimpleXML es una biblioteca de PHP que proporciona una manera fácil de trabajar con documentos XML (especialmente leer e iterar a través de datos XML).
- La única restricción es que el documento XML debe estar bien formado.

Análisis de XML utilizando un enfoque de procedimiento

```
// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = simplexml_load_string($xmlstr);

// Load an XML file
$library = simplexml_load_file('library.xml');

// You can load a local file path or a valid URL (if allow_url_fopen is set to "On" in php.ini
```

Analizar XML utilizando el enfoque OOP

```
// $isPathToFile: it informs the constructor that the 1st argument represents the path to a
file,
// rather than a string that contains the XML data itself.

// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);

// Load an XML file
$library = new SimpleXMLElement('library.xml', NULL, true);

// $isPathToFile: it informs the constructor that the first argument represents the path to a
file, rather than a string that contains the XML data itself.
```

Accediendo a los niños y atributos

- Cuando SimpleXML analiza un documento XML, convierte todos sus elementos XML o nodos en propiedades del objeto SimpleXMLElement resultante.
- Además, convierte los atributos XML en una matriz asociativa a la que se puede acceder desde la propiedad a la que pertenecen.

Cuando conoces sus nombres:

```
$library = new SimpleXMLElement('library.xml', NULL, true);
foreach ($library->book as $book) {
    echo $book['isbn'];
    echo $book->title;
```

```
    echo $book->author;
    echo $book->publisher;
}
```

- El principal inconveniente de este enfoque es que es necesario conocer los nombres de cada elemento y atributo en el documento XML.

Cuando no sabes sus nombres (o no quieres saberlos):

```
foreach ($library->children() as $child) {
    echo $child->getName();
    // Get attributes of this element
    foreach ($child->attributes() as $attr){
        echo ' ' . $attr->getName() . ':' . $attr;
    }
    // Get children
    foreach ($child->children() as $subchild){
        echo ' ' . $subchild->getName() . ':' . $subchild;
    }
}
```

Lea Análisis de HTML en línea: <https://riptutorial.com/es/php/topic/1032/analisis-de-html>

Capítulo 6: APCu

Introducción

APCu es un almacén de valor-clave de memoria compartida para PHP. La memoria se comparte entre los procesos PHP-FPM de la misma agrupación. Los datos almacenados persisten entre las solicitudes.

Examples

Almacenamiento y recuperación simples

`apcu_store` puede utilizarse para almacenar, `apcu_fetch` para recuperar valores:

```
$key = 'Hello';
$value = 'World';
apcu_store($key, $value);
print(apcu_fetch('Hello')); // 'World'
```

Almacenar información

`apcu_cache_info` proporciona información sobre la tienda y sus entradas:

```
print_r(apcu_cache_info());
```

Tenga en cuenta que invocar `apcu_cache_info()` sin límite devolverá los datos completos almacenados actualmente.

Para obtener solo los metadatos, use `apcu_cache_info(true)`.

Para obtener información sobre ciertas entradas de caché, utilice mejor `APCUIterator`.

Iterando sobre las entradas

El `APCUIterator` permite iterar sobre las entradas en el caché:

```
foreach (new APCUIterator() as $entry) {
    print_r($entry);
}
```

El iterador se puede inicializar con una expresión regular opcional para seleccionar solo entradas con claves coincidentes:

```
foreach (new APCUIterator($regex) as $entry) {
    print_r($entry);
}
```

La información sobre una sola entrada de caché se puede obtener a través de:

```
$key = '...';
$regex = '^' . preg_quote($key) . '$';
print_r((new APCUIIterator($regex))->current());
```

Lea APCu en línea: <https://riptutorial.com/es/php/topic/9894/apcu>

Capítulo 7: Aprendizaje automático

Observaciones

El tema utiliza PHP-ML para todos los algoritmos de aprendizaje automático. La instalación de la biblioteca se puede hacer usando

```
composer require php-ai/php-ml
```

El repositorio github para el mismo se puede encontrar [aquí](#).

También vale la pena señalar que los ejemplos que se dan son conjuntos de datos muy pequeños solo para fines de demostración. El conjunto de datos real debería ser más completo que eso.

Examples

Clasificación utilizando PHP-ML

La clasificación en aprendizaje automático es el problema que identifica a qué conjunto de categorías pertenece una nueva observación. La clasificación cae dentro de la categoría de Supervised Machine Learning .

Cualquier algoritmo que implementa clasificación se conoce como **clasificador**

Los clasificadores soportados en PHP-ML son

- SVC (Clasificación de vectores de soporte)
- k-vecinos más cercanos
- Ingenuo bayes

El método de `train` y `predict` es el mismo para todos los clasificadores. La única diferencia sería en el algoritmo subyacente utilizado.

SVC (Clasificación de vectores de soporte)

Antes de comenzar con la predicción de una nueva observación, debemos entrenar a nuestro clasificador. Considere el siguiente código

```
// Import library
use Phpml\Classification\SVC;
use Phpml\SupportVectorMachine\Kernel;

// Data for training classifier
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]]; // Training samples
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];
```

```
// Initialize the classifier
$classifier = new SVC(Kernel::LINEAR, $cost = 1000);
// Train the classifier
$classifier->train($samples, $labels);
```

El código es bastante sencillo. `$cost` utilizado anteriormente es una medida de cuánto queremos evitar errores de clasificación de cada ejemplo de capacitación. Por un valor menor de `$cost` puede obtener ejemplos mal clasificados. Por defecto se establece en 1.0

Ahora que hemos capacitado al clasificador, podemos comenzar a hacer algunas predicciones reales. Considera los siguientes códigos que tenemos para las predicciones.

```
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

El clasificador en el caso anterior puede tomar muestras sin clasificar y predice sus etiquetas. `predict` método de `predict` puede tomar una sola muestra, así como una serie de muestras.

k-vecinos más cercanos

El clasificador de este algoritmo toma dos parámetros y puede inicializarse como

```
$classifier = new KNearestNeighbors($neighbor_num=4);
$classifier = new KNearestNeighbors($neighbor_num=3, new Minkowski($lambda=4));
```

`$neighbor_num` es el número de vecinos más cercanos para escanear en el algoritmo `knn`, mientras que el segundo parámetro es la métrica de distancia, que por defecto en el primer caso sería `Euclidean`. Más sobre Minkowski se puede encontrar [aquí](#).

A continuación se muestra un breve ejemplo de cómo usar este clasificador.

```
// Training data
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize classifier
$classifier = new KNearestNeighbors();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

Clasificador NaiveBayes

`NaiveBayes Classifier` se basa en Bayes' theorem y no necesita ningún parámetro en el constructor.

El siguiente código demuestra una implementación de predicción simple

```
// Training data
$samples = [[5, 1, 1], [1, 5, 1], [1, 1, 5]];
$labels = ['a', 'b', 'c'];

// Initialize classifier
$classifier = new NaiveBayes();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 1, 1]); // return 'a'
$classifier->predict([[3, 1, 1], [1, 4, 1]]); // return ['a', 'b']
```

Caso práctico

Hasta ahora solo usamos matrices de enteros en todos nuestros casos, pero ese no es el caso en la vida real. Por lo tanto, permítame tratar de describir una situación práctica sobre cómo usar los clasificadores.

Supongamos que tiene una aplicación que almacena las características de las flores en la naturaleza. En aras de la simplicidad, podemos considerar el color y la longitud de los pétalos. Entonces, dos características serían usadas para entrenar nuestros datos. `color` es el más simple donde puede asignar un valor int a cada uno de ellos y para la longitud, puede tener un rango como $(0 \text{ mm}, 10 \text{ mm})=1$, $(10 \text{ mm}, 20 \text{ mm})=2$. Con los datos iniciales entrena a tu clasificador. Ahora, uno de sus usuarios necesita identificar el tipo de flor que crece en su patio trasero. Lo que hace es seleccionar el `color` de la flor y agrega la longitud de los pétalos. El clasificador en ejecución puede detectar el tipo de flor ("Etiquetas en el ejemplo anterior")

Regresión

En la clasificación utilizando `PHP-ML` asignamos etiquetas a una nueva observación. La regresión es casi la misma con la diferencia de que el valor de salida no es una etiqueta de clase sino un valor continuo. Es ampliamente utilizado para predicciones y previsiones. PHP-ML soporta los siguientes algoritmos de regresión

- Regresión de vectores de apoyo
- Regresión lineal de LeastSquares

La regresión tiene el mismo `train` y métodos de `predict` que los utilizados en la clasificación.

Regresión de vectores de apoyo

Esta es la versión de regresión para SVM (Máquina de vectores de soporte). El primer paso como en la clasificación es entrenar a nuestro modelo.

```

// Import library
use Phpml\Regression\SVR;
use Phpml\SupportVectorMachine\Kernel;

// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new SVR(Kernel::LINEAR);
// Train regression engine
$regression->train($samples, $targets);

```

En la regresión, `$targets` no son etiquetas de clase en lugar de clasificación. Este es uno de los factores diferenciadores para los dos. Después de entrenar nuestro modelo con los datos, podemos comenzar con las predicciones reales.

```
$regression->predict([64]) // return 4.03
```

Tenga en cuenta que las predicciones devuelven un valor fuera del objetivo.

Regresión lineal de LeastSquares

Este algoritmo utiliza el `least squares method` para aproximar la solución. Lo siguiente demuestra un simple código de entrenamiento y predicción.

```

// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new LeastSquares();
// Train engine
$regression->train($samples, $targets);
// Predict using trained engine
$regression->predict([64]); // return 4.06

```

PHP-ML también ofrece la opción de `Multiple Linear Regression`. Un código de ejemplo para el mismo puede ser el siguiente

```

$samples = [[73676, 1996], [77006, 1998], [10565, 2000], [146088, 1995], [15000, 2001],
[65940, 2000], [9300, 2000], [93739, 1996], [153260, 1994], [17764, 2002], [57000, 1998],
[15000, 2000]];
$targets = [2000, 2750, 15500, 960, 4400, 8800, 7100, 2550, 1025, 5900, 4600, 4400];

$regression = new LeastSquares();
$regression->train($samples, $targets);
$regression->predict([60000, 1996]) // return 4094.82

```

`Multiple Linear Regression` es particularmente útil cuando múltiples factores o rasgos identifican el resultado.

Caso práctico

Ahora tomemos una aplicación de regresión en el escenario de la vida real.

Supongamos que ejecuta un sitio web muy popular, pero el tráfico sigue cambiando. Desea una solución que pueda predecir la cantidad de servidores que necesita implementar en cualquier momento del tiempo. Supongamos por el hecho de que su proveedor de alojamiento le ofrece una API para generar servidores y cada servidor tarda 15 minutos en arrancar. En función de los datos anteriores de tráfico y regresión, puede predecir el tráfico que afectaría a su aplicación en cualquier momento. Gracias a ese conocimiento, puede iniciar un servidor 15 minutos antes de la oleada, lo que evita que su aplicación se desconecte.

Agrupación

La agrupación se trata de agrupar objetos similares juntos. Es ampliamente utilizado para el reconocimiento de patrones. Clustering se realiza bajo unsupervised machine learning , por lo tanto, no se necesita capacitación. PHP-ML tiene soporte para los siguientes algoritmos de clustering

- k-medios
- dbscan

k-medios

k-Means separa los datos en `n` grupos de igual varianza. Esto significa que debemos pasar un número `n` que sería el número de clústeres que necesitamos en nuestra solución. El siguiente código ayudará a traer más claridad

```
// Our data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

// Initialize clustering with parameter `n`
$kmmeans = new KMeans(3);
$kmmeans->cluster($samples); // return [0=>[[7, 8]], 1=>[[8, 7]], 2=>[[1,1]]]
```

Tenga en cuenta que la salida contiene 3 matrices porque ese era el valor de `n` en el constructor `KMeans` . También puede haber un segundo parámetro opcional en el constructor, que sería el `initialization method` . Por ejemplo considera

```
$kmmeans = new KMeans(4, KMeans::INIT_RANDOM);
```

`INIT_RANDOM` coloca un centroide completamente aleatorio al intentar determinar los clústeres. Pero solo para evitar que el centroide esté demasiado lejos de los datos, está limitado por los límites de espacio de los datos.

El `initialization method` predeterminado de `initialization method` constructor es `kmeans ++`, que selecciona el centroide de una manera inteligente para acelerar el proceso.

DBSCAN

A diferencia de KMeans , DBSCAN es un algoritmo de agrupamiento basado en densidad, lo que significa que no pasaremos n lo que determinaría la cantidad de clústeres que queremos en nuestro resultado. Por otro lado esto requiere dos parámetros para funcionar.

1. **\$ minSamples:** el número mínimo de objetos que deben estar presentes en un clúster
2. **\$ epsilon:** Cuál es la distancia máxima entre dos muestras para que se consideren en el mismo grupo.

Una muestra rápida para el mismo es la siguiente

```
// Our sample data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

$dbSCAN = new DBSCAN($epsilon = 2, $minSamples = 3);
$dbSCAN->cluster($samples); // return [0=>[[1, 1]], 1=>[[8, 7]]]
```

El código es bastante autoexplicativo. Una diferencia importante es que no hay forma de saber la cantidad de elementos en la matriz de salida en lugar de KMeans.

Caso práctico

Ahora echemos un vistazo al uso de agrupamiento en situaciones reales.

La agrupación en clústeres se utiliza ampliamente en el pattern recognition y data mining . Considera que tienes una aplicación de publicación de contenido. Ahora, para retener a sus usuarios, deben mirar el contenido que les encanta. Asumamos por simplicidad que si están en una página web específica durante más de un minuto y se van al fondo, les encanta ese contenido. Ahora, cada uno de sus contenidos tendrá un identificador único con él y el usuario también. Haga un cluster basado en eso y usted sabrá qué segmento de usuarios tiene un gusto similar al contenido. A su vez, esto podría usarse en el sistema de recomendaciones, donde puede asumir que si a algunos usuarios del mismo grupo les encanta el artículo, a otros les gustará, y eso puede mostrarse como recomendaciones en su aplicación.

Lea Aprendizaje automático en línea: <https://riptutorial.com/es/php/topic/5453/aprendizaje-automatico>

Capítulo 8: Arrays

Introducción

Una matriz es una estructura de datos que almacena un número arbitrario de valores en un solo valor. Una matriz en PHP es en realidad un mapa ordenado, donde mapa es un tipo que asocia valores a claves.

Sintaxis

- `$array = array ('Value1', 'Value2', 'Value3');` // Las teclas predeterminadas son 0, 1, 2, ...,
- `$array = array ('Value1', 'Value2',);` // coma final opcional
- `$array = array ('key1' => 'Value1', 'key2' => 'Value2',);` // Claves explícitas
- `$array = array ('key1' => 'Value1', 'Value2',);` // Array ([key1] => Value1 [1] => 'Value2')
- `$array = ['key1' => 'Value1', 'key2' => 'Value2',];` // PHP 5.4+ taquigrafía
- `$array [] = 'ValueX';` // Añadir 'ValueX' al final de la matriz
- `$array ['keyX'] = 'ValueX';` // Asignar 'valueX' a la tecla 'keyX'
- `$array += ['keyX' => 'valueX', 'keyY' => 'valueY'];` // Agregar / sobrescribir elementos en una matriz existente

Parámetros

Parámetro	Detalle
Llave	La clave es el identificador único y el índice de una matriz. Puede ser una <code>string</code> o un <code>integer</code> . Por lo tanto, las claves válidas serían <code>'foo'</code> , <code>'5'</code> , <code>10</code> , <code>'a2b'</code> , ...
Valor	Para cada <code>key</code> hay un valor correspondiente (de lo contrario, <code>null</code> y se emite un <i>aviso al acceder</i>). El valor no tiene restricciones en el tipo de entrada.

Observaciones

Ver también

- [Manipulando una sola matriz](#)
- [Ejecutando sobre una matriz](#)
- [Iteración de matriz](#)
- [Procesando múltiples matrices juntos](#)

Examples

Inicializando una matriz

Una matriz se puede inicializar vacía:

```
// An empty array
$foo = array();

// Shorthand notation available since PHP 5.4
$foo = [];
```

Una matriz se puede inicializar y preestablecer con valores:

```
// Creates a simple array with three strings
$fruit = array('apples', 'pears', 'oranges');

// Shorthand notation available since PHP 5.4
$fruit = ['apples', 'pears', 'oranges'];
```

Una matriz también se puede inicializar con índices personalizados (*también llamada matriz asociativa*) :

```
// A simple associative array
$fruit = array(
    'first' => 'apples',
    'second' => 'pears',
    'third' => 'oranges'
);

// Key and value can also be set as follows
$fruit['first'] = 'apples';

// Shorthand notation available since PHP 5.4
$fruit = [
    'first' => 'apples',
    'second' => 'pears',
    'third' => 'oranges'
];
```

Si la variable no se ha utilizado antes, PHP la creará automáticamente. Si bien es conveniente, esto podría hacer que el código sea más difícil de leer:

```
$foo[] = 1;      // Array( [0] => 1 )
$bar[][] = 2;    // Array( [0] => Array( [0] => 2 ) )
```

El índice usualmente continuará donde lo dejaste. PHP intentará usar cadenas numéricas como enteros:

```
$foo = [2 => 'apple', 'melon']; // Array( [2] => apple, [3] => melon )
$foo = ['2' => 'apple', 'melon']; // same as above
$foo = [2 => 'apple', 'this is index 3 temporarily', '3' => 'melon']; // same as above! The
last entry will overwrite the second!
```

Para inicializar una matriz con un tamaño fijo, puede usar [SplFixedArray](#) :

```
$array = new SplFixedArray(3);

$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
$array[3] = 4; // RuntimeException

// Increase the size of the array to 10
$array->setSize(10);
```

Nota: una matriz creada con [SplFixedArray](#) tiene una huella de memoria reducida para grandes conjuntos de datos, pero las claves deben ser enteros.

Para inicializar una matriz con un tamaño dinámico pero con `n` elementos no vacíos (por ejemplo, un marcador de posición), puede usar un bucle de la siguiente manera:

```
$myArray = array();
$sizeOfMyArray = 5;
$fill = 'placeholder';

for ($i = 0; $i < $sizeOfMyArray; $i++) {
    $myArray[] = $fill;
}

// print_r($myArray); results in the following:
// Array ( [0] => placeholder [1] => placeholder [2] => placeholder [3] => placeholder [4] =>
placeholder )
```

Si todos sus marcadores de posición son iguales, también puede crearlo utilizando la función [array_fill\(\)](#) :

`array array_fill (int $ start_index, int $ num, mixed $ value)`

Esto crea y devuelve una matriz con `num` entradas de `value`, llaves a partir de `start_index`.

Nota: Si el `start_index` es negativo, comenzará con el índice negativo y continuará desde 0 para los siguientes elementos.

```
$a = array_fill(5, 6, 'banana'); // Array ( [5] => banana, [6] => banana, ..., [10] => banana)
$b = array_fill(-2, 4, 'pear'); // Array ( [-2] => pear, [0] => pear, ..., [2] => pear)
```

Conclusión: con `array_fill()` estás más limitado por lo que realmente puedes hacer. El bucle es más flexible y le abre una gama más amplia de oportunidades.

Cuando quiera que una matriz se llene con un rango de números (por ejemplo, 1-4), puede agregar cada elemento a una matriz o usar la función `range()`:

rango de matriz (mezcla \$ inicio, mezcla \$ final [, número \$ paso = 1])

Esta función crea una matriz que contiene un rango de elementos. Los primeros dos parámetros son necesarios, donde establecen los puntos de inicio y final del rango (inclusive). El tercer parámetro es opcional y define el tamaño de los pasos que se están tomando. Al crear un `range` de 0 a 4 con un `stepsize` de 1, la matriz resultante consistirá de los siguientes elementos: 0, 1, 2, 3 y 4. Si el tamaño del paso se incrementa a 2 (es decir, `range(0, 4, 2)`), la matriz resultante sería: 0, 2 y 4.

```
$array = [];
$array_with_range = range(1, 4);

for ($i = 1; $i <= 4; $i++) {
    $array[] = $i;
}

print_r($array); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
print_r($array_with_range); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`range` puede trabajar con enteros, flotadores, valores booleanos (que se convierten en enteros) y cadenas. Sin embargo, se debe tener precaución al usar flotantes como argumentos debido al problema de precisión de punto flotante.

Comprobar si existe la clave

Utilice `array_key_exists()` O `isset()` O `!empty()` `isset()` `!empty()`:

```
$map = [
    'foo' => 1,
    'bar' => null,
    'foobar' => '',
];

array_key_exists('foo', $map); // true
isset($map['foo']); // true
!empty($map['foo']); // true

array_key_exists('bar', $map); // true
isset($map['bar']); // false
!empty($map['bar']); // false
```

Tenga en cuenta que `isset()` trata un elemento de valor `null` como no existente. Mientras que `!empty()` hace lo mismo para cualquier elemento que sea `false` (usando una comparación débil; por ejemplo, `null`, '' y 0 se tratan como falso por `!empty()`). While `isset($map['foobar'])`; es `true` `!empty($map['foobar'])` es `false`. Esto puede llevar a errores (por ejemplo, es fácil olvidar que la cadena '' se trata como falsa), por lo que el uso de `!empty()` menudo está mal visto.

Tenga en cuenta también que `isset()` y `!empty()` `isset()` funcionarán (y devolverán `false`) si `$map` no está definido en absoluto. Esto los hace un tanto propensos a usar errores:

```
// Note "long" vs "lang", a tiny typo in the variable name.  
$my_array_with_a_long_name = ['foo' => true];  
array_key_exists('foo', $my_array_with_a_lang_name); // shows a warning  
isset($my_array_with_a_lang_name['foo']); // returns false
```

También puede consultar matrices ordinales:

```
$ord = ['a', 'b']; // equivalent to [0 => 'a', 1 => 'b']  
  
array_key_exists(0, $ord); // true  
array_key_exists(2, $ord); // false
```

Tenga en cuenta que `isset()` tiene mejor rendimiento que `array_key_exists()` ya que esta última es una función y la primera es una construcción de lenguaje.

También puedes usar `key_exists()`, que es un alias para `array_key_exists()`.

Comprobando si existe un valor en la matriz

La función `in_array()` devuelve `true` si existe un elemento en una matriz.

```
$fruits = ['banana', 'apple'];  
  
$foo = in_array('banana', $fruits);  
// $foo value is true  
  
$bar = in_array('orange', $fruits);  
// $bar value is false
```

También puede usar la función `array_search()` para obtener la clave de un elemento específico en una matriz.

```
$userdb = ['Sandra Shush', 'Stefanie McMohn', 'Michael'];  
$pos = array_search('Stefanie McMohn', $userdb);  
if ($pos !== false) {  
    echo "Stefanie McMohn found at $pos";  
}
```

PHP 5.x 5.5

En PHP 5.5 y `array_column()` posteriores, puede usar `array_column()` junto con `array_search()`.

Esto es particularmente útil para [verificar si existe un valor en una matriz asociativa](#):

```
$userdb = [  
    [  
        "uid" => '100',  
        "name" => 'Sandra Shush',  
        "url" => 'urlof100',
```

```

],
[
    "uid" => '5465',
    "name" => 'Stefanie Mcmohn',
    "pic_square" => 'urlof100',
],
[
    "uid" => '40489',
    "name" => 'Michael',
    "pic_square" => 'urlof40489',
]
];
$key = array_search(40489, array_column($userdb, 'uid'));

```

Validando el tipo de matriz

La función `is_array()` devuelve true si una variable es una matriz.

```

$integer = 1337;
$array = [1337, 42];

is_array($integer); // false
is_array($array); // true

```

Puede escribir sugerir el tipo de matriz en una función para imponer un tipo de parámetro; Pasar cualquier otra cosa resultará en un error fatal.

```
function foo (array $array) { /* $array is an array */ }
```

También puede utilizar la función `gettype()`.

```

$integer = 1337;
$array = [1337, 42];

gettype($integer) === 'array'; // false
gettype($array) === 'array'; // true

```

Interfaces ArrayAccess e Iterador

Otra característica útil es acceder a sus colecciones de objetos personalizados como matrices en PHP. Hay dos interfaces disponibles en PHP (> = 5.0.0) núcleo para apoyar esta: `ArrayAccess` y `Iterator`. El primero le permite acceder a sus objetos personalizados como matriz.

ArrayAccess

Supongamos que tenemos una clase de usuario y una tabla de base de datos que almacena a todos los usuarios. Nos gustaría crear una clase `UserCollection` que:

1. Permítanos dirigirnos a ciertos usuarios por su nombre de usuario identificador único
2. realizar operaciones básicas (no todas las CRUD, pero al menos Crear, Recuperar y Eliminar) en nuestra colección de usuarios

Considere la siguiente fuente (en lo sucesivo, usaremos la sintaxis de creación de matriz corta [] disponible desde la versión 5.4):

```
class UserCollection implements ArrayAccess {
    protected $_conn;

    protected $_requiredParams = ['username', 'password', 'email'];

    public function __construct() {
        $config = new Configuration();

        $connectionParams = [
            //your connection to the database
        ];

        $this->_conn = DriverManager::getConnection($connectionParams, $config);
    }

    protected function _getByUsername($username) {
        $ret = $this->_conn->executeQuery('SELECT * FROM `User` WHERE `username` IN (?)',
            [$username]
        )->fetch();

        return $ret;
    }

    // START of methods required by ArrayAccess interface
    public function offsetExists($offset) {
        return (bool) $this->_getByUsername($offset);
    }

    public function offsetGet($offset) {
        return $this->_getByUsername($offset);
    }

    public function offsetSet($offset, $value) {
        if (!is_array($value)) {
            throw new \Exception('value must be an Array');
        }

        $passed = array_intersect(array_values($this->_requiredParams), array_keys($value));
        if (count($passed) < count($this->_requiredParams)) {
            throw new \Exception('value must contain at least the following params: ' .
                implode(', ', $this->_requiredParams));
        }
        $this->_conn->insert('User', $value);
    }

    public function offsetUnset($offset) {
        if (!is_string($offset)) {
            throw new \Exception('value must be the username to delete');
        }
        if (!$this->offsetGet($offset)) {
            throw new \Exception('user not found');
        }
        $this->_conn->delete('User', ['username' => $offset]);
    }
    // END of methods required by ArrayAccess interface
}
```

Entonces podemos :

```
$users = new UserCollection();

var_dump(empty($users['testuser']), isset($users['testuser']));
$users['testuser'] = ['username' => 'testuser',
                     'password' => 'testpassword',
                     'email'     => 'test@test.com'];
var_dump(empty($users['testuser']), isset($users['testuser']), $users['testuser']);
unset($users['testuser']);
var_dump(empty($users['testuser']), isset($users['testuser']));
```

que generará lo siguiente, asumiendo que no hubo un `testuser` antes de lanzar el código:

```
bool(true)
bool(false)
bool(false)
bool(true)
array(17) {
    ["username"]=>
        string(8) "testuser"
    ["password"]=>
        string(12) "testpassword"
    ["email"]=>
        string(13) "test@test.com"
}
bool(true)
bool(false)
```

IMPORTANTE: no se llama a `offsetExists` cuando verifica la existencia de una clave con la función `array_key_exists`. Entonces el siguiente código dará como resultado `false` dos veces:

```
var_dump(array_key_exists('testuser', $users));
$users['testuser'] = ['username' => 'testuser',
                     'password' => 'testpassword',
                     'email'     => 'test@test.com'];
var_dump(array_key_exists('testuser', $users));
```

Iterador

Extendamos nuestra clase desde arriba con algunas funciones de la interfaz del `Iterator` para permitir la iteración sobre ella con `foreach` y `while`.

Primero, debemos agregar una propiedad que contenga nuestro índice actual de iterador, vamos a agregarla a las propiedades de la clase como `$_position`:

```
// iterator current position, required by Iterator interface methods
protected $_position = 1;
```

Segundo, agreguemos la interfaz `Iterator` a la lista de interfaces implementadas por nuestra clase:

```
class UserCollection implements ArrayAccess, Iterator {
```

A continuación, agregue lo requerido por las funciones de la interfaz en sí:

```
// START of methods required by Iterator interface
public function current () {
    return $this->_getById($this->_position);
}
public function key () {
    return $this->_position;
}
public function next () {
    $this->_position++;
}
public function rewind () {
    $this->_position = 1;
}
public function valid () {
    return null !== $this->_getById($this->_position);
}
// END of methods required by Iterator interface
```

Entonces, en general, aquí está la fuente completa de la clase que implementa ambas interfaces. Tenga en cuenta que este ejemplo no es perfecta, ya que los identificadores en la base de datos pueden no ser secuencial, pero esto fue escrito sólo para darle la idea principal: se puede hacer frente a sus colecciones de objetos de cualquier manera posible mediante la implementación de **ArrayAccess** y **Iterator interfaces**:

```
class UserCollection implements ArrayAccess, Iterator {
    // iterator current position, required by Iterator interface methods
    protected $_position = 1;

    // <add the old methods from the last code snippet here>

    // START of methods required by Iterator interface
    public function current () {
        return $this->_getById($this->_position);
    }
    public function key () {
        return $this->_position;
    }
    public function next () {
        $this->_position++;
    }
    public function rewind () {
        $this->_position = 1;
    }
    public function valid () {
        return null !== $this->_getById($this->_position);
    }
    // END of methods required by Iterator interface
}
```

y un bucle **foreach** a través de todos los objetos de usuario:

```
foreach ($users as $user) {
    var_dump($user['id']);
}
```

lo que producirá algo así como

```
string(2) "1"
string(2) "2"
string(2) "3"
string(2) "4"
...
```

Creando una matriz de variables

```
$username = 'Hadibut';
$email = 'hadibut@example.org';

$variables = compact('username', 'email');
// $variables is now ['username' => 'Hadibut', 'email' => 'hadibut@example.org']
```

Este método se usa a menudo en marcos para pasar una matriz de variables entre dos componentes.

Lea Arrays en línea: <https://riptutorial.com/es/php/topic/204/arrays>

Capítulo 9: Asegurate recuerdame

Introducción

He estado buscando en este tema durante algún tiempo hasta que encontré esta publicación <https://stackoverflow.com/a/17266448/4535386> de ircmaxell, creo que merece más exposición.

Examples

"Mantenerme conectado" - el mejor enfoque

Almacenar la cookie con tres partes.

```
function onLogin($user) {
    $token = GenerateRandomToken(); // generate a token, should be 128 - 256 bit
    storeTokenForUser($user, $token);
    $cookie = $user . ':' . $token;
    $mac = hash_hmac('sha256', $cookie, SECRET_KEY);
    $cookie .= ':' . $mac;
    setcookie('rememberme', $cookie);
}
```

Luego, para validar:

```
function rememberMe() {
    $cookie = isset($_COOKIE['rememberme']) ? $_COOKIE['rememberme'] : '';
    if ($cookie) {
        list ($user, $token, $mac) = explode(':', $cookie);
        if (!hash_equals(hash_hmac('sha256', $user . ':' . $token, SECRET_KEY), $mac)) {
            return false;
        }
        $usertoken = fetchTokenByUserName($user);
        if (hash_equals($usertoken, $token)) {
            logUserIn($user);
        }
    }
}
```

Lea Asegurate recuerdame en línea: <https://riptutorial.com/es/php/topic/10664/asegurate-recuerdame>

Capítulo 10: Autenticación HTTP

Introducción

En este tema vamos a hacer un script de autenticación HTTP-Header.

Examples

Autenticación simple

TENGA EN CUENTA: SOLO PONGA ESTE CÓDIGO EN EL TÍTULO DE LA PÁGINA, DE OTRA MANERA NO FUNCIONARÁ!

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
}
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
$user = $_SERVER['PHP_AUTH_USER']; //Lets save the information
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
$pass = $_SERVER['PHP_AUTH_PW']; //Save the password(optionally add encryption) !
?>
//You html page
```

Lea Autenticación HTTP en línea: <https://riptutorial.com/es/php/topic/8059/autenticacion-http>

Capítulo 11: BC Math (calculadora binaria)

Introducción

La calculadora binaria se puede utilizar para calcular con números de cualquier tamaño y precisión hasta 2147483647-1 decimales, en formato de cadena. La calculadora binaria es más precisa que el cálculo flotante de PHP.

Sintaxis

- string bcadd (string \$ left_operand, string \$ right_operand [, int \$ scale = 0])
- int bccomp (string \$ left_operand, string \$ right_operand [, int \$ scale = 0])
- string bcddiv (string \$ left_operand, string \$ right_operand [, int \$ scale = 0])
- string bcmmod (string \$ left_operand, string \$ modulus)
- string bcmul (string \$ left_operand, string \$ right_operand [, int \$ scale = 0])
- string bcpowmod (string \$ left_operand, string \$ right_operand, string \$ modulus [, int \$ scale = 0])
- bool bcscale (int \$ scale)
- string bcsqrt (string \$ operand [, int \$ scale = 0])
- string bbsub (string \$ left_operand, string \$ right_operand [, int \$ scale = 0])

Parámetros

bcadd	Suma dos números de precisión arbitrarios.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bccomp	Compara dos números de precisión arbitrarios.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal que se utilizará en la comparación.
bcddiv	Divide dos números de precisión arbitrarios.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.

bcadd	Suma dos números de precisión arbitrarios.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bcmod	Obtener el módulo de un número de precisión arbitrario.
left_operand	El operando izquierdo, como una cuerda.
modulus	El módulo, como una cuerda.
bcmul	Multiplica dos números de precisión arbitrarios.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bcpow	Elevar un número de precisión arbitrario a otro.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bcpowmod	Elevar un número de precisión arbitrario a otro, reducido por un módulo específico.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.
modulus	El módulo, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
Escala	Establecer el parámetro de escala predeterminado para todas las funciones matemáticas bc.
scale	El factor de escala.
bcsqrt	Obtener la raíz cuadrada de un número de precisión arbitraria.
operand	El operando, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del

bcadd	Suma dos números de precisión arbitrarios.
	lugar decimal en el resultado.
bcscale	Resta un número de precisión arbitrario de otro.
left_operand	El operando izquierdo, como una cuerda.
right_operand	El operando correcto, como una cuerda.
scale	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.

Observaciones

Para todas las funciones de BC, si el parámetro de `scale` no está establecido, el valor predeterminado es 0, lo que hará que todas las operaciones sean operaciones con enteros.

Examples

Comparación entre BCMath y operaciones aritméticas flotantes

bcadd vs float + float

```
var_dump('10' + '-9.99');           // float(0.00999999999998)
var_dump(10 + -9.99);              // float(0.00999999999998)
var_dump(10.00 + -9.99);           // float(0.00999999999998)
var_dump(bcadd('10', '-9.99', 20)); // string(22) "0.01000000000000000000000000000000"
```

bcscale vs float-float

```
var_dump('10' - '9.99');           // float(0.00999999999998)
var_dump(10 - 9.99);              // float(0.00999999999998)
var_dump(10.00 - 9.99);           // float(0.00999999999998)
var_dump(bcscale('10', '9.99', 20)); // string(22) "0.01000000000000000000000000000000"
```

bcmul vs int * int

```
var_dump('5.00' * '2.00');          // float(10)
var_dump(5.00 * 2.00);              // float(10)
var_dump(bcmul('5.0', '2', 20));    // string(4) "10.0"
var_dump(bcmul('5.000', '2.00', 20)); // string(8) "10.00000"
var_dump(bcmul('5', '2', 20));      // string(2) "10"
```

bcmul vs float * float

```
var_dump('1.6767676767' * '1.6767676767');           // float(2.8115498416259)
var_dump(1.6767676767 * 1.6767676767);             // float(2.8115498416259)
var_dump bcmul('1.6767676767', '1.6767676767', 20)); // string(22) "2.81154984162591572289"
```

bcddiv vs float / float

```
var_dump('10' / '3.01');                  // float(3.3222591362126)
var_dump(10 / 3.01);                     // float(3.3222591362126)
var_dump(10.00 / 3.01);                  // float(3.3222591362126)
var_dump bcddiv('10', '3.01', 20));      // string(22) "3.32225913621262458471"
```

Uso de bcmath para leer / escribir un binario largo en un sistema de 32 bits

En los sistemas de 32 bits, los enteros mayores que `0x7FFFFFFF` no pueden almacenarse primitivamente, mientras que los enteros entre `0x0000000080000000` y `0x7FFFFFFFFFFFFFFF` pueden almacenarse primitivamente en sistemas de 64 bits pero no en sistemas de 32 bits (`signed long long`). Sin embargo, dado que los sistemas de 64 bits y muchos otros lenguajes admiten el almacenamiento de enteros `signed long long`, a veces es necesario almacenar este rango de enteros en el valor exacto. Hay varias formas de hacerlo, como crear una matriz con dos números o convertir el número entero en su forma decimal legible para el hombre. Esto tiene varias ventajas, como la conveniencia de presentar al usuario y la capacidad de manipularlo directamente con bcmath.

Los métodos de `pack` / `unpack` se pueden usar para convertir entre bytes binarios y la forma decimal de los números (ambos de tipo `string`, pero uno es binario y el otro es ASCII), pero siempre intentarán convertir la cadena ASCII en un formato de 32 bits. `int` en sistemas de 32 bits. El siguiente fragmento de código proporciona una alternativa:

```
/** Use pack("J") or pack("p") for 64-bit systems */
function writeLong(string $ascii) : string {
    if(bccomp($ascii, "0") === -1) { // if $ascii < 0
        // 18446744073709551616 is equal to (1 << 64)
        // remember to add the quotes, or the number will be parsed as a float literal
        $ascii = bcadd($ascii, "18446744073709551616");
    }

    // "n" is big-endian 16-bit unsigned short. Use "v" for small-endian.
    return pack("n", bcmod(bcddiv($ascii, "281474976710656"), "65536")) .
        pack("n", bcmod(bcddiv($ascii, "4294967296"), "65536")) .
        pack("n", bcddiv($ascii, "65536"), "65536")) .
        pack("n", bcmod($ascii, "65536"));
}

function readLong(string $binary) : string {
    $result = "0";
    $result = bcadd($result, unpack("n", substr($binary, 0, 2)));
    $result = bcmul($result, "65536");
```

```
$result = bcadd($result, unpack("n", substr($binary, 2, 2)));
$result = bcmul($result, "65536");
$result = bcadd($result, unpack("n", substr($binary, 4, 2)));
$result = bcmul($result, "65536");
$result = bcadd($result, unpack("n", substr($binary, 6, 2)));

// if $binary is a signed long long
// 9223372036854775808 is equal to (1 << 63) (note that this expression actually does not
work even on 64-bit systems)
if(bccomp($result, "9223372036854775808") != -1) { // if $result >= 9223372036854775807
    $result = bcsub($result, "18446744073709551616"); // $result -= (1 << 64)
}
return $result;
}
```

Lea BC Math (calculadora binaria) en línea: <https://riptutorial.com/es/php/topic/8550/bc-math--calculadora-binaria->

Capítulo 12: Bucles

Introducción

Los bucles son un aspecto fundamental de la programación. Permiten a los programadores crear código que se repite para un número determinado de repeticiones o *iteraciones*. El número de iteraciones puede ser explícito (6 iteraciones, por ejemplo), o continuar hasta que se cumpla alguna condición ('hasta que el infierno se congele').

Este tema cubre los diferentes tipos de bucles, sus declaraciones de control asociadas y sus posibles aplicaciones en PHP.

Sintaxis

- `para` (contador de inicio; contador de prueba; contador de incremento) {/ * código * /}
- `foreach` (matriz como valor) {/ * código * /}
- `foreach` (matriz como clave => valor) {/ * código * /}
- `while` (condición) {/ * código * /}
- `hacer` {/ * código * /} `while` (condición);
- `anyloop` {continuar; }
- `anyloop` {[`anyloop` ...]} {`continue int`; }}
- `anyloop` {descanso; }
- `anyloop` {[`anyloop` ...]} {`break int`; }}

Observaciones

A menudo es útil ejecutar el mismo bloque de código o varias veces. En lugar de copiar y pegar, los bucles de sentencias casi iguales proporcionan un mecanismo para ejecutar el código un número específico de veces y recorrer estructuras de datos. PHP soporta los siguientes cuatro tipos de bucles:

- `for`
- `while`
- `do...while`
- `foreach`

Para controlar estos bucles, `continue` y las declaraciones de `break` están disponibles.

Examples

para

La instrucción `for` se utiliza cuando sabe cuántas veces desea ejecutar una instrucción o un bloque de instrucciones.

El inicializador se utiliza para establecer el valor de inicio para el contador del número de iteraciones de bucle. Se puede declarar una variable aquí para este propósito y es tradicional nombrarla `$i`.

El siguiente ejemplo itera 10 veces y muestra números del 0 al 9.

```
for ($i = 0; $i <= 9; $i++) {  
    echo $i, ',';  
}  
  
# Example 2  
for ($i = 0; ; $i++) {  
    if ($i > 9) {  
        break;  
    }  
    echo $i, ',';  
}  
  
# Example 3  
$i = 0;  
for (; ; ) {  
    if ($i > 9) {  
        break;  
    }  
    echo $i, ',';  
    $i++;  
}  
  
# Example 4  
for ($i = 0, $j = 0; $i <= 9; $j += $i, print $i. ',', $i++);
```

La salida esperada es:

```
0,1,2,3,4,5,6,7,8,9,
```

para cada

La instrucción `foreach` se utiliza para hacer un ciclo a través de matrices.

Para cada iteración, el valor del elemento de la matriz actual se asigna a la variable `$value` y el puntero de la matriz se mueve uno y en la siguiente iteración se procesará el siguiente elemento.

El siguiente ejemplo muestra los elementos en la matriz asignada.

```
$list = ['apple', 'banana', 'cherry'];  
  
foreach ($list as $value) {  
    echo "I love to eat {$value}. ";  
}
```

La salida esperada es:

```
I love to eat apple. I love to eat banana. I love to eat cherry.
```

También puede acceder a la clave / índice de un valor utilizando `foreach`:

```
foreach ($list as $key => $value) {  
    echo $key . ":" . $value . " "  
}  
  
//Outputs - 0:apple 1:banana 2:cherry
```

Por defecto, `$value` es una copia del valor en `$list`, por lo que los cambios realizados dentro del bucle no se reflejarán en `$list` después.

```
foreach ($list as $value) {  
    $value = $value . " pie";  
}  
echo $list[0]; // Outputs "apple"
```

Para modificar la matriz dentro del bucle `foreach`, use el operador `&` para asignar `$value` por referencia. Es importante `unset` la variable luego para que reusar `$value` otro lugar no sobrescriba la matriz.

```
foreach ($list as &$value) { // Or foreach ($list as $key => &$value) {  
    $value = $value . " pie";  
}  
unset($value);  
echo $list[0]; // Outputs "apple pie"
```

También puede modificar los elementos de la matriz dentro del bucle `foreach` haciendo referencia a la clave de la matriz del elemento actual.

```
foreach ($list as $key => $value) {  
    $list[$key] = $value . " pie";  
}  
echo $list[0]; // Outputs "apple pie"
```

descanso

La palabra clave `break` termina inmediatamente el bucle actual.

De manera similar a la instrucción `continue`, una `break` detiene la ejecución de un bucle. Sin embargo, a diferencia de una instrucción de `continue`, la `break` provoca la terminación inmediata del bucle y *no ejecuta* la instrucción condicional nuevamente.

```
$i = 5;  
while(true) {  
    echo 120/$i.PHP_EOL;  
    $i -= 1;  
    if ($i == 0) {  
        break;  
    }  
}
```

Este código producirá

```
24  
30  
40  
60  
120
```

pero no ejecutará el caso donde `$i` es 0, lo que resultaría en un error fatal debido a la división por 0.

La instrucción `break` también puede usarse para romper varios niveles de bucles. Tal comportamiento es muy útil cuando se ejecutan bucles anidados. Por ejemplo, para copiar una matriz de cadenas en una cadena de salida, eliminando cualquier `#` símbolo, hasta que la cadena de salida tenga exactamente 160 caracteres

```
$output = "";  
$inputs = array(  
    "#soblessed #throwbackthursday",  
    "happy tuesday",  
    "#nofilter",  
    /* more inputs */  
)  
foreach($inputs as $input) {  
    for($i = 0; $i < strlen($input); $i += 1) {  
        if ($input[$i] == '#') continue;  
        $output .= $input[$i];  
        if (strlen($output) == 160) break 2;  
    }  
    $output .= ' ';
```

El comando `break 2` termina inmediatamente la ejecución de los bucles interno y externo.

hacer ... mientras

La instrucción `do...while` ejecutará un bloque de código al menos una vez, luego repetirá el ciclo siempre que la condición sea verdadera.

El siguiente ejemplo incrementará el valor de `$i` al menos una vez, y continuará incrementando la variable `$i` siempre que tenga un valor inferior a 25;

```
$i = 0;  
do {  
    $i++;  
} while($i < 25);  
  
echo 'The final value of i is: ', $i;
```

La salida esperada es:

```
The final value of i is: 25
```

continuar

La palabra clave `continue` detiene la iteración actual de un bucle pero no termina el bucle.

Al igual que la instrucción `break`, la instrucción `continue` está situada dentro del cuerpo del bucle. Cuando se ejecuta, la instrucción `continue` hace que la ejecución salte inmediatamente al bucle condicional.

En el siguiente ejemplo, el bucle imprime un mensaje basado en los valores de una matriz, pero omite un valor especificado.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    if ($value == 'banana') {
        continue;
    }
    echo "I love to eat {$value} pie.".PHP_EOL;
}
```

La salida esperada es:

```
I love to eat apple pie.
I love to eat cherry pie.
```

La instrucción de `continue` también se puede usar para continuar inmediatamente la ejecución a un nivel externo de un bucle especificando la cantidad de niveles de bucle para saltar. Por ejemplo, considere datos como

Fruta	Color	Costo
manzana	rojo	1
Plátano	Amarillo	7
Cereza	rojo	2
Uva	Verde	4

Para hacer solo pasteles de fruta que cuestan menos de 5.

```
$data = [
    [ "Fruit" => "Apple", "Color" => "Red", "Cost" => 1 ],
    [ "Fruit" => "Banana", "Color" => "Yellow", "Cost" => 7 ],
    [ "Fruit" => "Cherry", "Color" => "Red", "Cost" => 2 ],
    [ "Fruit" => "Grape", "Color" => "Green", "Cost" => 4 ]
];

foreach($data as $fruit) {
    foreach($fruit as $key => $value) {
        if ($key == "Cost" && $value >= 5) {
            continue 2;
        }
    }
}
```

```
    /* make a pie */
}
}
```

Cuando se ejecuta la instrucción `continue 2`, la ejecución salta inmediatamente a `$data as $fruit` continúa con el bucle externo y omite todos los demás códigos (incluido el condicional en el bucle interno).

mientras

La instrucción `while` ejecutará un bloque de código siempre y cuando la expresión de prueba sea verdadera.

Si la expresión de prueba es verdadera, entonces se ejecutará el bloque de código. Después de que el código se haya ejecutado, la expresión de prueba se evaluará nuevamente y el bucle continuará hasta que se encuentre que la expresión de prueba es falsa.

El siguiente ejemplo itera hasta que la suma alcanza 100 antes de terminar.

```
$i = true;
$sum = 0;

while ($i) {
    if ($sum === 100) {
        $i = false;
    } else {
        $sum += 10;
    }
}
echo 'The sum is: ', $sum;
```

La salida esperada es:

```
The sum is: 100
```

Lea Bucles en línea: <https://riptutorial.com/es/php/topic/2213/bucles>

Capítulo 13: Buffer de salida

Parámetros

Función	Detalles
ob_start ()	Inicia el búfer de salida, cualquier salida colocada después de esto se capturará y no se mostrará
ob_get_contents ()	Devuelve todo el contenido capturado por ob_start ()
ob_end_clean ()	Vacía el búfer de salida y lo desactiva para el nivel de anidamiento actual
ob_get_clean ()	Activa tanto ob_get_contents () como ob_end_clean()
ob_get_level ()	Devuelve el nivel de anidamiento actual del búfer de salida.
ob_flush ()	Descargue el búfer de contenido y envíelo al navegador sin finalizar el búfer
ob_implicit_flush ()	Habilita el vaciado implícito después de cada llamada de salida.
ob_end_flush ()	Vacie el búfer de contenido y envíelo al navegador, también finalizando el búfer

Examples

Uso básico obteniendo contenido entre buffers y clearing

El búfer de salida le permite almacenar cualquier contenido textual (Texto, HTML) en una variable y enviarlo al navegador como una pieza al final de su script. Por defecto, php envía su contenido como lo interpreta.

```
<?php

// Turn on output buffering
ob_start();

// Print some output to the buffer (via php)
print 'Hello ';

// You can also `step out` of PHP
?>
<em>World</em>
<?php
```

```

// Return the buffer AND clear it
$content = ob_get_clean();

// Return our buffer and then clear it
# $content = ob_get_contents();
# $did_clear_buffer = ob_end_clean();

print($content);

#> "Hello <em>World</em>"
```

Cualquier contenido `ob_start()` entre `ob_start()` y `ob_get_clean()` será capturado y colocado en la variable `$content`.

Al llamar a `ob_get_clean()` `ob_get_contents()` tanto `ob_get_contents()` como `ob_end_clean()`.

Buffers de salida anidados

Puede anidar buffers de salida y obtener el nivel para que proporcionen contenido diferente utilizando la función `ob_get_level()`.

```

<?php

$i = 1;
$output = null;

while( $i <= 5 ) {
    // Each loop, creates a new output buffering `level`
    ob_start();
    print "Current nest level: " . ob_get_level() . "\n";
    $i++;
}

// We're at level 5 now
print 'Ended up at level: ' . ob_get_level() . PHP_EOL;

// Get clean will `pop` the contents of the top most level (5)
$output .= ob_get_clean();
print $output;

print 'Popped level 5, so we now start from 4' . PHP_EOL;

// We're now at level 4 (we pop'ed off 5 above)

// For each level we went up, come back down and get the buffer
while( $i > 2 ) {
    print "Current nest level: " . ob_get_level() . "\n";
    echo ob_get_clean();
    $i--;
}
```

Salidas:

```

Current nest level: 1
Current nest level: 2
Current nest level: 3
```

```
Current nest level: 4
Current nest level: 5
Ended up at level: 5
Popped level 5, so we now start from 4
Current nest level: 4
Current nest level: 3
Current nest level: 2
Current nest level: 1
```

Capturando el buffer de salida para reutilizarlo más tarde.

En este ejemplo, tenemos una matriz que contiene algunos datos.

\$items_li_html búfer de salida en \$items_li_html y lo usamos dos veces en la página.

```
<?php

// Start capturing the output
ob_start();

$items = ['Home', 'Blog', 'FAQ', 'Contact'];

foreach($items as $item):

// Note we're about to step "out of PHP land"
?>
<li><?php echo $item ?></li>
<?php
// Back in PHP land
endforeach;

// $items_lists contains all the HTML captured by the output buffer
$items_li_html = ob_get_clean();
?>

<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
    <?php echo $items_li_html ?>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
    <?php echo $items_li_html ?>
</ul>
```

Guarde el código anterior en un archivo `output_buffer.php` y ejecútelo a través de `php output_buffer.php`.

Debería ver los 2 elementos de lista que creamos anteriormente con los mismos elementos de lista que generamos en PHP usando el búfer de salida:

```
<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
    <li>Home</li>
    <li>Blog</li>
    <li>FAQ</li>
```

```

<li>Contact</li>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
    <li>Home</li>
    <li>Blog</li>
    <li>FAQ</li>
    <li>Contact</li>
</ul>

```

Ejecutando buffer de salida antes de cualquier contenido.

```

ob_start();

$user_count = 0;
foreach( $users as $user ) {
    if( $user['access'] != 7 ) { continue; }
    ?>
    <li class="users user-<?php echo $user['id']; ?>">
        <a href="<?php echo $user['link']; ?>">
            <?php echo $user['name'] ?>
        </a>
    </li>
    <?php
        $user_count++;
    ?>
    $users_html = ob_get_clean();

if( !$user_count ) {
    header('Location: /404.php');
    exit();
}
?>
<html>
<head>
    <title>Level 7 user results (<?php echo $user_count; ?>)</title>
</head>

<body>
<h2>We have a total of <?php echo $user_count; ?> users with access level 7</h2>
<ul class="user-list">
    <?php echo $users_html; ?>
</ul>
</body>
</html>

```

En este ejemplo, asumimos que `$users` son una matriz multidimensional, y lo hacemos en bucle para encontrar a todos los usuarios con un nivel de acceso de 7.

Si no hay resultados, redirigimos a una página de error.

Estamos utilizando el búfer de salida aquí porque estamos activando un redireccionamiento de `header()` basado en el resultado del bucle

Uso del búfer de salida para almacenar contenidos en un archivo, útil para

informes, facturas, etc.

```
<?php
ob_start();
?>
<html>
<head>
    <title>Example invoice</title>
</head>
<body>
<h1>Invoice #0000</h1>
<h2>Cost: £15,000</h2>
...
</body>
</html>
<?php
$html = ob_get_clean();

$handle = fopen('invoices/example-invoice.html', 'w');
fwrite($handle, $html);
fclose($handle);
```

Este ejemplo toma el documento completo y lo escribe en un archivo, no lo imprime en el navegador, pero lo hace usando `echo $html;`

Procesando el búfer a través de una devolución de llamada

Puede aplicar cualquier tipo de procesamiento adicional a la salida pasando un llamable a `ob_start()`.

```
<?php
function clearAllWhiteSpace($buffer) {
    return str_replace(array("\n", "\t", ' '), '', $buffer);
}

ob_start('clearAllWhiteSpace');
?>
<h1>Lorem Ipsum</h1>

<p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada fames
ac turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p>

<h2>Header Level 2</h2>

<ol>
    <li>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li>
    <li>Aliquam tincidunt mauris eu risus.</li>
</ol>

<?php
/* Output will be flushed and processed when script ends or call
   ob_end_flush();
*/
```

Salida:

```
<h1>Lorem Ipsum</h1><p><strong>Pellentesque habitant morbit tristique</strong> senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, sem. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusamus et iusto odio dignissimos
```

Transmitir salida al cliente

```
/**  
 * Enables output buffer streaming. Calling this function  
 * immediately flushes the buffer to the client, and any  
 * subsequent output will be sent directly to the client.  
 */  
function _stream() {  
    ob_implicit_flush(true);  
    ob_end_flush();  
}
```

Uso típico y razones para usar ob_start

`ob_start` es especialmente útil cuando tienes redirecciones en tu página. Por ejemplo, el siguiente código no funcionará:

```
Hello!  
<?php  
    header("Location: somepage.php");  
?>
```

El error que se dará es algo como: `headers already sent by <xxxx> on line <xxxx>`.

Para solucionar este problema, debe escribir algo como esto al comienzo de su página:

```
<?php  
    ob_start();  
?>
```

Y algo como esto al final de tu página:

```
<?php  
    ob_end_flush();  
?>
```

Esto almacena todo el contenido generado en un búfer de salida y lo muestra de una sola vez. Por lo tanto, si tiene llamadas de redirección en su página, éstas se activarán antes de que se envíe cualquier dato, eliminando la posibilidad de que se produzcan errores en los `headers already sent`.

Lea Buffer de salida en línea: <https://riptutorial.com/es/php/topic/541/buffer-de-salida>

Capítulo 14: Cache

Observaciones

Instalación

Puedes instalar memcache usando pecl

```
pecl install memcache
```

Examples

Caché utilizando memcache

Memcache es un sistema de almacenamiento en caché de objetos distribuidos y utiliza `key-value` para almacenar datos pequeños. Antes de comenzar a llamar código de `Memcache` a PHP, debe asegurarse de que esté instalado. Eso se puede hacer usando el método `class_exists` en php. Una vez que se valida que el módulo está instalado, comienza a conectarse a la instancia del servidor memcache.

```
if (class_exists('Memcache')) {
    $cache = new Memcache();
    $cache->connect('localhost', 11211);
} else {
    print "Not connected to cache server";
}
```

Esto validará que los controladores php de Memcache estén instalados y se conectarán a la instancia del servidor memcache que se ejecuta en localhost.

Memcache se ejecuta como un demonio y se llama **memcached**

En el ejemplo anterior solo nos conectamos a una sola instancia, pero también puede conectarse a múltiples servidores usando

```
if (class_exists('Memcache')) {
    $cache = new Memcache();
    $cache->addServer('192.168.0.100', 11211);
    $cache->addServer('192.168.0.101', 11211);
}
```

Tenga en cuenta que, en este caso, a diferencia de la conexión, no habrá ninguna conexión activa hasta que intente almacenar o recuperar un valor.

En el almacenamiento en caché hay tres operaciones importantes que deben implementarse

1. **Almacenar datos:** agregar nuevos datos al servidor memcached

2. **Obtener datos:** obtener datos del servidor memcached
3. **Eliminar datos:** eliminar datos ya existentes del servidor memcached

Almacenamiento de datos

`$cache` o objeto de clase memcached tiene un método de `set` que toma una clave, valor y tiempo para guardar el valor para (ttl).

```
$cache->set($key, $value, 0, $ttl);
```

Aquí `$ ttl` o `time to live` es el tiempo en segundos que desea que memcache almacene el par en el servidor.

Obtener datos

`$cache` o objeto de clase memcached tiene un método de `get` que toma una clave y devuelve el valor correspondiente.

```
$value = $cache->get($key);
```

En caso de que no haya un valor establecido para la clave, se devolverá **nulo**.

Borrar datos

A veces es posible que tenga que eliminar algún valor de caché. `$cache` instancia de `$cache` o memcache tiene un método de `delete` que puede usarse para el mismo.

```
$cache->delete($key);
```

Pequeño escenario para el almacenamiento en caché

Asumamos un simple blog. Tendrá múltiples publicaciones en la página de destino que se obtendrán de la base de datos con cada carga de página. Para reducir las consultas de SQL podemos usar memcached para almacenar en caché las publicaciones. Aquí hay una implementación muy pequeña.

```
if (class_exists('Memcache')) {  
    $cache = new Memcache();  
    $cache->connect('localhost', 11211);  
    if (($data = $cache->get('posts')) != null) {  
        // Cache hit  
        // Render from cache  
    } else {  
        // Cache miss  
    }  
}
```

```

    // Query database and save results to database
    // Assuming $posts is array of posts retrieved from database
    $cache->set('posts', $posts, 0, $ttl);
}
} else {
    die("Error while connecting to cache server");
}

```

Caché utilizando caché APC

El caché de PHP alternativo (APC) es un caché de código de operación gratuito y abierto para PHP. Su objetivo es proporcionar un marco gratuito, abierto y robusto para almacenar en caché y optimizar el código intermedio de PHP.

instalación

```

sudo apt-get install php-apc
sudo /etc/init.d/apache2 restart

```

Añadir caché:

```

apc_add ($key, $value , $ttl);
$key = unique cache key
$value = cache value
$ttl = Time To Live;

```

Eliminar caché:

```

apc_delete($key);

```

Ejemplo de Set Cache:

```

if (apc_exists($key)) {
    echo "Key exists: ";
    echo apc_fetch($key);
} else {
    echo "Key does not exist";
    apc_add ($key, $value , $ttl);
}

```

Rendimiento :

APC es casi **5 veces** más rápido que Memcached.

Lea Cache en línea: <https://riptutorial.com/es/php/topic/5470/cache>

Capítulo 15: Cierre

Examples

Uso básico de un cierre.

Un **cierre** es el equivalente de PHP de una función anónima, por ejemplo. Una función que no tiene nombre. Incluso si eso no es técnicamente correcto, el comportamiento de un cierre sigue siendo el mismo que el de una función, con algunas características adicionales.

Un cierre no es más que un objeto de la clase Closure que se crea al declarar una función sin nombre. Por ejemplo:

```
<?php

$myClosure = function() {
    echo 'Hello world!';
};

$myClosure(); // Shows "Hello world!"
```

Tenga en cuenta que `$myClosure` es una instancia de `Closure` para que esté al tanto de lo que realmente puede hacer con ella (consulte <http://fr2.php.net/manual/en/class.closure.php>)

El caso clásico que se necesita un cierre es cuando se tiene que dar una `callable` a una función, por ejemplo `usort`.

Aquí hay un ejemplo donde una matriz está ordenada por el número de hermanos de cada persona:

```
<?php

$data = [
    [
        'name' => 'John',
        'nbrOfSiblings' => 2,
    ],
    [
        'name' => 'Stan',
        'nbrOfSiblings' => 1,
    ],
    [
        'name' => 'Tom',
        'nbrOfSiblings' => 3,
    ]
];

usort($data, function($e1, $e2) {
    if ($e1['nbrOfSiblings'] == $e2['nbrOfSiblings']) {
        return 0;
    }
})
```

```

        return $e1['nbrOfSiblings'] < $e2['nbrOfSiblings'] ? -1 : 1;
    });

var_dump($data); // Will show Stan first, then John and finally Tom

```

Utilizando variables externas

Es posible, dentro de un cierre, utilizar una variable externa con el **uso** especial de palabras clave. Por ejemplo:

```

<?php

$quantity = 1;

$calculator = function($number) use($quantity) {
    return $number + $quantity;
};

var_dump($calculator(2)); // Shows "3"

```

Puedes ir más lejos creando cierres "dinámicos". Es posible crear una función que devuelva una calculadora específica, dependiendo de la cantidad que desee agregar. Por ejemplo:

```

<?php

function createCalculator($quantity) {
    return function($number) use($quantity) {
        return $number + $quantity;
    };
}

$calculator1 = createCalculator(1);
$calculator2 = createCalculator(2);

var_dump($calculator1(2)); // Shows "3"
var_dump($calculator2(2)); // Shows "4"

```

Encuadernación de cierre básico.

Como se vio anteriormente, un cierre no es más que una instancia de la clase Closure, y se pueden invocar diferentes métodos en ellos. Uno de ellos es `bindTo`, que, dado un cierre, devolverá uno nuevo que está vinculado a un objeto dado. Por ejemplo:

```

<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyName)

```

```

    {
        $this->property = $propertyName;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"

```

Cierre de encuadernación y alcance.

Consideremos este ejemplo:

```

<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyName)
    {
        $this->property = $propertyName;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"

```

Intente cambiar la visibilidad de la `property` a `protected` o `private`. Recibes un error fatal que indica que no tienes acceso a esta propiedad. De hecho, incluso si el cierre se ha vinculado al objeto, el alcance en el que se invoca el cierre no es el necesario para tener ese acceso. Para eso es el segundo argumento de `bindTo`.

La única forma de acceder a una propiedad si es `private` es que se accede desde un ámbito que lo permita, es decir. El alcance de la clase. En el ejemplo de código anterior, el alcance no se ha especificado, lo que significa que el cierre se ha invocado en el mismo ámbito que el utilizado donde se creó el cierre. Vamos a cambiar eso:

```

<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    private $property; // $property is now private
}

```

```

public function __construct($PropertyValue)
{
    $this->property = $PropertyValue;
}
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance, MyClass::class);

$myBoundClosure(); // Shows "Hello world!"

```

Como acabo de decir, si este segundo parámetro no se utiliza, el cierre se invoca en el mismo contexto que el utilizado donde se creó el cierre. Por ejemplo, un cierre creado dentro de una clase de método que se invoca en un contexto de objeto tendrá el mismo alcance que el método:

```

<?php

class MyClass
{
    private $property;

    public function __construct ($PropertyValue)
    {
        $this->property = $PropertyValue;
    }

    public function getDisplayer()
    {
        return function() {
            echo $this->property;
        };
    }
}

$myInstance = new MyClass('Hello world!');

$displayer = $myInstance->getDisplayer();
$displayer(); // Shows "Hello world!"

```

Encuadernación de un cierre para una llamada.

Desde PHP7 , es posible vincular un cierre solo para una llamada, gracias al método de `call` . Por ejemplo:

```

<?php

class MyClass
{
    private $property;

    public function __construct ($PropertyValue)
    {
        $this->property = $PropertyValue;
    }
}

$myClosure = function() {

```

```

        echo $this->property;
    };

$myInstance = new MyClass('Hello world!');

$myClosure->call($myInstance); // Shows "Hello world!"

```

A diferencia del método `bindTo`, no hay que preocuparse por el alcance. El alcance utilizado para esta llamada es el mismo que el utilizado para acceder o invocar una propiedad de `$myInstance`.

Utilizar cierres para implementar patrón observador.

En general, un observador es una clase con un método específico que se llama cuando ocurre una acción en el objeto observado. En ciertas situaciones, los cierres pueden ser suficientes para implementar el patrón de diseño del observador.

Aquí hay un ejemplo detallado de tal implementación. Primero declaremos una clase cuyo propósito es notificar a los observadores cuando se cambia su propiedad.

```

<?php

class ObservedStuff implements SplSubject
{
    protected $property;
    protected $observers = [];

    public function attach(SplObserver $observer)
    {
        $this->observers[] = $observer;
        return $this;
    }

    public function detach(SplObserver $observer)
    {
        if (false !== $key = array_search($observer, $this->observers, true)) {
            unset($this->observers[$key]);
        }
    }

    public function notify()
    {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function getProperty()
    {
        return $this->property;
    }

    public function setProperty($property)
    {
        $this->property = $property;
        $this->notify();
    }
}

```

Luego, declaremos la clase que representará a los diferentes observadores.

```
<?php

class NamedObserver implements SplObserver
{
    protected $name;
    protected $closure;

    public function __construct(Closure $closure, $name)
    {
        $this->closure = $closure->bindTo($this, $this);
        $this->name = $name;
    }

    public function update(SplSubject $subject)
    {
        $closure = $this->closure;
        $closure($subject);
    }
}
```

Por fin probemos esto:

```
<?php

$o = new ObservedStuff;

$observer1 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$observer2 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$o->attach(new NamedObserver($observer1, 'Observer1'))
->attach(new NamedObserver($observer2, 'Observer2'));

$o->setProperty('Hello world!');
// Shows:
// Observer1 has been notified! New property value: Hello world!
// Observer2 has been notified! New property value: Hello world!
```

Tenga en cuenta que este ejemplo funciona porque los observadores comparten la misma naturaleza (ambos son "observadores nombrados").

Lea Cierre en línea: <https://riptutorial.com/es/php/topic/2634/cierre>

Capítulo 16: Clase de fecha y hora

Examples

getTimestamp

`getTimeStamp` es una representación de unix de un objeto `datetime`.

```
$date = new DateTime();
echo $date->getTimestamp();
```

Esto pondrá una indicación de número entero en los segundos que han transcurrido desde las 00:00:00 UTC del jueves 1 de enero de 1970.

Establece la fecha

`setDate` establece la fecha en un objeto `DateTime`.

```
$date = new DateTime();
$date->setDate(2016, 7, 25);
```

este ejemplo establece que la fecha será el veinticinco de julio de 2015 y producirá el siguiente resultado:

```
2016-07-25 17:52:15.819442
```

Agregar o restar intervalos de fecha

Podemos usar la clase `DateInterval` para agregar o restar algún intervalo en un objeto `DateTime`.

Vea el ejemplo a continuación, donde estamos agregando un intervalo de 7 días e imprimiendo un mensaje en la pantalla:

```
$now = new DateTime(); // empty argument returns the current date
$interval = new DateInterval('P7D'); //this object represents a 7 days interval
$lastDay = $now->add($interval); //this will return a DateTime object
$formattedLastDay = $lastDay->format('Y-m-d'); //this method formats the DateTime object and
returns a String
echo "Samara says: Seven Days. You'll be happy on $formattedLastDay.;"
```

Esto saldrá (funcionando el 1 de agosto de 2016):

Samara dice: siete días. Estarás feliz el 2016-08-08.

Podemos usar el método `sub` de una manera similar para restar fechas

```
$now->sub($interval);
```

```
echo "Samara says: Seven Days. You were happy last on $formatedLastDay.";
```

Esto saldrá (funcionando el 1 de agosto de 2016):

Samara dice: siete días. Estuviste feliz el ultimo 2016-07-25.

Crea DateTime desde un formato personalizado

PHP es capaz de analizar [una serie de formatos de fecha](#). Si desea analizar un formato no estándar, o si desea que su código `DateTime::createFromFormat` explícitamente el formato que se utilizará, puede usar el método estático `DateTime::createFromFormat`:

Estilo orientado a objetos

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = DateTime::createFromFormat($format, $time);
```

Estilo procesal

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = date_create_from_format($format, $time);
```

Imprimir DateTimes

PHP 4+ proporciona un método, formato que convierte un objeto DateTime en una cadena con un formato deseado. Según el manual de PHP, esta es la función orientada a objetos:

```
public string DateTime::format ( string $format )
```

La función date () toma un parámetro: un formato, que es una cadena

Formato

El formato es una cadena y utiliza caracteres únicos para definir el formato:

- **Y** : representación de cuatro dígitos del año (ej.: 2016)
- **y** : representación de dos dígitos del año (p. ej.: 16)
- **m** : mes, como un número (01 a 12)
- **M** : mes, como tres letras (enero, febrero, marzo, etc.)
- **j** : día del mes, sin ceros iniciales (1 a 31)
- **D** : día de la semana, como tres letras (lunes, martes, miércoles, etc.)
- **h** : hora (formato de 12 horas) (01 a 12)
- **H** : hora (formato de 24 horas) (00 a 23)
- **A** : ya sea AM o PM
- **i** : minuto, con ceros iniciales (00 a 59)

- **s** : segundo, con ceros iniciales (00 a 59)
- La lista completa se puede encontrar [aquí](#).

Uso

Estos caracteres se pueden usar en varias combinaciones para mostrar los tiempos en prácticamente cualquier formato. Aquí hay unos ejemplos:

```
$date = new DateTime('2000-05-26T13:30:20'); /* Friday, May 26, 2000 at 1:30:20 PM */

$date->format("H:i");
/* Returns 13:30 */

$date->format("H i s");
/* Returns 13 30 20 */

$date->format("h:i:s A");
/* Returns 01:30:20 PM */

$date->format("j/m/Y");
/* Returns 26/05/2000 */

$date->format("D, M j 'y - h:i A");
/* Returns Fri, May 26 '00 - 01:30 PM */
```

Procesal

El formato procesal es similar:

Orientado a objetos

```
$date->format($format)
```

Equivalente de procedimiento

```
date_format($date, $format)
```

Cree una versión inmutable de DateTime desde Mutable antes de PHP 5.6

Para crear `\DateTimeImmutable` en PHP 5.6+ use:

```
\DateTimeImmutable::createFromMutable($concrete);
```

Antes de PHP 5.6 puedes usar:

```
\DateTimeImmutable::createFromFormat(\DateTime::ISO8601, $mutable->format(\DateTime::ISO8601),
$mutable->getTimezone());
```

Lea Clase de fecha y hora en línea: <https://riptutorial.com/es/php/topic/3684/clase-de-fecha-y-hora>

Capítulo 17: Clases y objetos

Introducción

Las clases y los objetos se utilizan para hacer que su código sea más eficiente y menos repetitivo al agrupar tareas similares.

Una clase se usa para definir las acciones y la estructura de datos utilizada para construir objetos. Los objetos se construyen utilizando esta estructura predefinida.

Sintaxis

- `class <ClassName> [extends <ParentClassName>] [implements <Interface1> [, <Interface2>, ...] { }` // Declaración de clase
- `interface <InterfaceName> [extends <ParentInterface1> [, <ParentInterface2>, ...] { }` // Declaración de interfaz
- `use <Trait1> [, <Trait2>, ...];` // Usar rasgos
- `[public | protected | private] [static] $<varName>;` // Declaración de atributo
- `const <CONST_NAME>;` // Declaración constante
- `[public | protected | private] [static] function <methodName>([args...]) { }` // Declaración de método

Observaciones

Clases y componentes de interfaz

Las clases pueden tener propiedades, constantes y métodos.

- **Las propiedades** mantienen variables en el alcance del objeto. Pueden inicializarse en la declaración, pero solo si contienen un valor primitivo.
- **Las constantes** deben inicializarse en la declaración y solo pueden contener un valor primitivo. Los valores constantes se fijan en el momento de la compilación y no pueden asignarse en el tiempo de ejecución.
- **Los métodos** deben tener un cuerpo, incluso uno vacío, a menos que el método se declare abstracto.

```
class Foo {  
    private $foo = 'foo'; // OK  
    private $baz = array(); // OK  
    private $bar = new Bar(); // Error!  
}
```

Las interfaces no pueden tener propiedades, pero pueden tener constantes y métodos.

- **Las constantes de interfaz** deben inicializarse en la declaración y solo pueden contener un

valor primitivo. Los valores constantes se fijan en el momento de la compilación y no pueden asignarse en el tiempo de ejecución.

- Los **métodos de interfaz** no tienen cuerpo.

```
interface FooBar {  
    const FOO_VALUE = 'bla';  
    public function doAnything();  
}
```

Examples

Interfaces

Introducción

Las interfaces son definiciones de las clases de API públicas que deben implementarse para satisfacer la interfaz. Funcionan como "contratos", especificando **lo que hace** un conjunto de subclases, pero **no cómo** lo hacen.

La definición de interfaz es muy parecida a la definición de clase, cambiando la `class` palabra clave a `interface`:

```
interface Foo {  
}
```

Las interfaces pueden contener métodos y / o constantes, pero no atributos. Las constantes de interfaz tienen las mismas restricciones que las constantes de clase. Los métodos de interfaz son **implícitamente abstractos**:

```
interface Foo {  
    const BAR = 'BAR';  
  
    public function doSomething($param1, $param2);  
}
```

Nota: las interfaces **no deben** declarar constructores o destructores, ya que estos son detalles de implementación en el nivel de clase.

Realización

Cualquier clase que necesite implementar una interfaz debe hacerlo usando la palabra clave `implements`. Para hacerlo, la clase debe proporcionar una implementación para cada método declarado en la interfaz, respetando la misma firma.

Una sola clase **puede** implementar más de una interfaz a la vez.

```

interface Foo {
    public function doSomething($param1, $param2);
}

interface Bar {
    public function doAnotherThing($param1);
}

class Baz implements Foo, Bar {
    public function doSomething($param1, $param2) {
        // ...
    }

    public function doAnotherThing($param1) {
        // ...
    }
}

```

Cuando las clases abstractas implementan interfaces, no necesitan implementar todos los métodos. Cualquier método no implementado en la clase base debe ser implementado por la clase concreta que lo extiende:

```

abstract class AbstractBaz implements Foo, Bar {
    // Partial implementation of the required interface...
    public function doSomething($param1, $param2) {
        // ...
    }
}

class Baz extends AbstractBaz {
    public function doAnotherThing($param1) {
        // ...
    }
}

```

Observe que la realización de la interfaz es una característica heredada. Al extender una clase que implementa una interfaz, no es necesario volver a declararla en la clase concreta, porque está implícita.

Nota: Antes de PHP 5.3.9, una clase no podía implementar dos interfaces que especificaban un método con el mismo nombre, ya que causaría ambigüedad. Las versiones más recientes de PHP lo permiten siempre que los métodos duplicados tengan la misma firma [1].

Herencia

Al igual que las clases, es posible establecer una relación de herencia entre interfaces, utilizando la misma palabra clave `extends`. La principal diferencia es que se permite la herencia múltiple para las interfaces:

```
interface Foo {
```

```

}

interface Bar {

}

interface Baz extends Foo, Bar {
}

```

Ejemplos

En el siguiente ejemplo tenemos una interfaz de ejemplo simple para un vehículo. Los vehículos pueden ir hacia adelante y hacia atrás.

```

interface VehicleInterface {
    public function forward();

    public function reverse();

    ...
}

class Bike implements VehicleInterface {
    public function forward() {
        $this->pedal();
    }

    public function reverse() {
        $this->backwardSteps();
    }

    protected function pedal() {
        ...
    }

    protected function backwardSteps() {
        ...
    }

    ...
}

class Car implements VehicleInterface {
    protected $gear = 'N';

    public function forward() {
        $this->setGear(1);
        $this->pushPedal();
    }

    public function reverse() {
        $this->setGear('R');
        $this->pushPedal();
    }

    protected function setGear($gear) {

```

```

        $this->gear = $gear;
    }

    protected function pushPedal() {
        ...
    }

    ...
}

```

Luego creamos dos clases que implementan la interfaz: Bicicleta y Coche. Bicicletas y automóviles internamente son muy diferentes, pero ambos son vehículos, y deben implementar los mismos métodos públicos que proporciona VehicleInterface.

La tipografía permite que los métodos y funciones soliciten interfaces. Supongamos que tenemos una clase de estacionamiento, que contiene vehículos de todo tipo.

```

class ParkingGarage {
    protected $vehicles = [];

    public function addVehicle(VehicleInterface $vehicle) {
        $this->vehicles[] = $vehicle;
    }
}

```

Debido a que `addVehicle` requiere un `$vehicle` del tipo `VehicleInterface` no una implementación concreta, podemos ingresar Bicicletas y Autos, que el `ParkingGarage` puede manipular y usar.

Constantes de clase

Las constantes de clase proporcionan un mecanismo para mantener valores fijos en un programa. Es decir, proporcionan una forma de asignar un nombre (y una comprobación de tiempo de compilación asociada) a un valor como `3.14` o `"Apple"`. Las constantes de clase solo se pueden definir con la palabra clave `const`: la función de `definir` no se puede usar en este contexto.

Como ejemplo, puede ser conveniente tener una representación abreviada del valor de π en todo el programa. Una clase con valores `const` proporciona una forma sencilla de mantener dichos valores.

```

class MathValues {
    const PI = M_PI;
    const PHI = 1.61803;
}

$area = MathValues::PI * $radius * $radius;

```

Se puede acceder a las constantes de clase utilizando el operador de dos puntos dobles (denominado operador de resolución de alcance) en una clase, al igual que las variables estáticas. Sin embargo, a diferencia de las variables estáticas, las constantes de clase tienen sus valores fijos en el momento de la compilación y no se pueden reasignar a (por ejemplo, `MathValues::PI = 7` produciría un error fatal).

Las constantes de clase también son útiles para definir cosas internas de una clase que pueden necesitar cambiar más adelante (pero no cambian con la frecuencia suficiente para justificar el almacenamiento en, por ejemplo, una base de datos). Podemos hacer referencia a esto internamente utilizando el resolutor de alcance `self` (que funciona tanto en implementaciones estáticas como en instancias)

```
class Labor {  
    /** How long, in hours, does it take to build the item? */  
    const LABOR_UNITS = 0.26;  
    /** How much are we paying employees per hour? */  
    const LABOR_COST = 12.75;  
  
    public function getLaborCost($number_units) {  
        return (self::LABOR_UNITS * self::LABOR_COST) * $number_units;  
    }  
}
```

Las constantes de clase solo pueden contener valores escalares en versiones <5.6

A partir de PHP 5.6 podemos usar expresiones con constantes, lo que significa que las declaraciones matemáticas y las cadenas con concatenación son constantes aceptables

```
class Labor {  
    /** How much are we paying employees per hour? Hourly wages * hours taken to make */  
    const LABOR_COSTS = 12.75 * 0.26;  
  
    public function getLaborCost($number_units) {  
        return self::LABOR_COSTS * $number_units;  
    }  
}
```

A partir de PHP 7.0, las constantes declaradas con `define` ahora pueden contener matrices.

```
define("BAZ", array('baz'));
```

Las constantes de clase son útiles para algo más que almacenar conceptos matemáticos. Por ejemplo, si prepara una tarta, puede ser conveniente tener una clase de `Pie` capaz de tomar diferentes tipos de fruta.

```
class Pie {  
    protected $fruit;  
  
    public function __construct($fruit) {  
        $this->fruit = $fruit;  
    }  
}
```

Entonces podemos usar la clase `Pie` como tal

```
$pie = new Pie("strawberry");
```

El problema que surge aquí es que, al crear una instancia de la clase `Pie`, no se proporciona una

guía sobre los valores aceptables. Por ejemplo, al hacer una tarta "boysenberry", podría escribirse incorrectamente "boisenberry". Además, podríamos no apoyar una tarta de ciruela. En su lugar, sería útil tener una lista de tipos de frutas aceptables ya definidas en algún lugar en el que tendría sentido buscarlas. Di una clase llamada `Fruit`:

```
class Fruit {  
    const APPLE = "apple";  
    const STRAWBERRY = "strawberry";  
    const BOYSENBERRY = "boysenberry";  
}  
  
$pie = new Pie(Fruit::STRAWBERRY);
```

Enumerar los valores aceptables como constantes de clase proporciona una sugerencia valiosa sobre los valores aceptables que acepta un método. También asegura que las faltas de ortografía no puedan superar el compilador. Si bien la `new Pie('aple')` y la `new Pie('apple')` son aceptables para el compilador, la `new Pie(Fruit::APLE)` producirá un error de compilación.

Finalmente, el uso de constantes de clase significa que el valor real de la constante puede modificarse en un solo lugar, y cualquier código que use la constante tiene automáticamente los efectos de la modificación.

Si bien el método más común para acceder a una constante de clase es `MyClass::CONSTANT_NAME`, también se puede acceder a él mediante:

```
echo MyClass::CONSTANT;  
  
$classname = "MyClass";  
echo $classname::CONSTANT; // As of PHP 5.3.0
```

Las constantes de clase en PHP se denominan convencionalmente todas en mayúsculas con guiones bajos como separadores de palabras, aunque cualquier nombre de etiqueta válido se puede usar como un nombre de constante de clase.

A partir de PHP 7.1, las constantes de clase ahora se pueden definir con visibilidades diferentes del alcance público predeterminado. Esto significa que tanto las constantes protegidas como las privadas pueden definirse ahora para evitar que las constantes de clase se filtren innecesariamente en el ámbito público (consulte [Método y visibilidad de la propiedad](#)). Por ejemplo:

```
class Something {  
    const PUBLIC_CONST_A = 1;  
    public const PUBLIC_CONST_B = 2;  
    protected const PROTECTED_CONST = 3;  
    private const PRIVATE_CONST = 4;  
}
```

definir constantes de clase vs

Aunque esta es una construcción válida:

```
function bar() { return 2; };

define('BAR', bar());
```

Si intentas hacer lo mismo con las constantes de clase, obtendrás un error:

```
function bar() { return 2; };

class Foo {
    const BAR = bar(); // Error: Constant expression contains invalid operations
}
```

Pero puedes hacer:

```
function bar() { return 2; };

define('BAR', bar());

class Foo {
    const BAR = BAR; // OK
}
```

Para más información, ver [constantes en el manual](#).

Usando :: class para recuperar el nombre de la clase

PHP 5.5 introdujo la `::class` sintaxis de `::class` para recuperar el nombre completo de la clase, teniendo en cuenta el alcance del espacio de nombres y las declaraciones de `use`.

```
namespace foo;
use bar\Bar;
echo json_encode(Bar::class); // "bar\\Bar"
echo json_encode(Foo::class); // "foo\\Foo"
echo json_encode(\Foo::class); // "Foo"
```

Lo anterior funciona incluso si las clases ni siquiera están definidas (es decir, este fragmento de código funciona solo).

Esta sintaxis es útil para funciones que requieren un nombre de clase. Por ejemplo, se puede usar con `class_exists` para verificar que una clase existe. No se generarán errores, independientemente del valor de retorno en este fragmento:

```
class_exists(ThisClass\Will\NeverBe\Loaded::class, false);
```

Enlace estático tardío

En PHP 5.3 y versiones posteriores, puede utilizar el enlace estático tardío para controlar desde qué clase de clase de propiedad o método se llama. Se agregó para superar el problema inherente con el resolutor `self::` scope. Toma el siguiente código

```
class Horse {
    public static function whatToSay() {
        echo 'Neigh!';
    }

    public static function speak() {
        self::whatToSay();
    }
}

class MrEd extends Horse {
    public static function whatToSay() {
        echo 'Hello Wilbur!';
    }
}
```

Usted esperaría que la clase `MrEd` anule la función principal `whatToSay()`. Pero cuando corremos esto obtenemos algo inesperado.

```
Horse::speak(); // Neigh!
MrEd::speak(); // Neigh!
```

El problema es que `self::whatToSay();` solo puede referirse a la clase `Horse`, lo que significa que no obedece a `MrEd`. Si cambiamos al resolutor `static::` scope, no tenemos este problema. Este nuevo método le dice a la clase que obedezca la instancia que lo llama. Así obtenemos la herencia que estamos esperando.

```
class Horse {
    public static function whatToSay() {
        echo 'Neigh!';
    }

    public static function speak() {
        static::whatToSay(); // Late Static Binding
    }
}

Horse::speak(); // Neigh!
MrEd::speak(); // Hello Wilbur!
```

Clases abstractas

Una clase abstracta es una clase que no puede ser instanciada. Las clases abstractas pueden definir métodos abstractos, que son métodos sin cuerpo, solo una definición:

```
abstract class MyAbstractClass {
    abstract public function doSomething($a, $b);
}
```

Las clases abstractas deben extenderse por una clase secundaria que luego puede proporcionar la implementación de estos métodos abstractos.

El propósito principal de una clase como esta es proporcionar un tipo de plantilla que permita a las clases de niños heredar, "forzando" una estructura a adherirse. Vamos a elaborar sobre esto con un ejemplo:

En este ejemplo estaremos implementando una interfaz `Worker`. Primero definimos la interfaz:

```
interface Worker {  
    public function run();  
}
```

Para facilitar el desarrollo de futuras implementaciones de `Worker`, crearemos una clase de trabajo abstracta que ya proporciona el método `run()` desde la interfaz, pero especifica algunos métodos abstractos que deben ser completados por cualquier clase secundaria:

```
abstract class AbstractWorker implements Worker {  
    protected $pdo;  
    protected $logger;  
  
    public function __construct(PDO $pdo, Logger $logger) {  
        $this->pdo = $pdo;  
        $this->logger = $logger;  
    }  
  
    public function run() {  
        try {  
            $this->setMemoryLimit($this->getMemoryLimit());  
            $this->logger->log("Preparing main");  
            $this->prepareMain();  
            $this->logger->log("Executing main");  
            $this->main();  
        } catch (Throwable $e) {  
            // Catch and rethrow all errors so they can be logged by the worker  
            $this->logger->log("Worker failed with exception: {$e->getMessage()}");  
            throw $e;  
        }  
    }  
  
    private function setMemoryLimit($memoryLimit) {  
        ini_set('memory_limit', $memoryLimit);  
        $this->logger->log("Set memory limit to $memoryLimit");  
    }  
  
    abstract protected function getMemoryLimit();  
  
    abstract protected function prepareMain();  
  
    abstract protected function main();  
}
```

En primer lugar, hemos proporcionado un método abstracto `getMemoryLimit()`. Cualquier clase que se extienda desde `AbstractWorker` debe proporcionar este método y devolver su límite de memoria. El `AbstractWorker` luego establece el límite de memoria y lo registra.

En segundo lugar, `AbstractWorker` llama a los `prepareMain()` y `main()`, después de registrar que se han llamado.

Finalmente, todas estas llamadas de método se han agrupado en un bloque `try - catch`. Entonces, si alguno de los métodos abstractos definidos por la clase secundaria produce una excepción, capturaremos esa excepción, la registraremos y la volveremos a realizar. Esto evita que todas las clases secundarias tengan que implementar esto ellos mismos.

Ahora definamos una clase secundaria que se extiende desde `AbstractWorker`:

```
class TransactionProcessorWorker extends AbstractWorker {
    private $transactions;

    protected function getMemoryLimit() {
        return "512M";
    }

    protected function prepareMain() {
        $stmt = $this->pdo->query("SELECT * FROM transactions WHERE processed = 0 LIMIT 500");
        $stmt->execute();
        $this->transactions = $stmt->fetchAll();
    }

    protected function main() {
        foreach ($this->transactions as $transaction) {
            // Could throw some PDO or MYSQL exception, but that is handled by the
            AbstractWorker
            $stmt = $this->pdo->query("UPDATE transactions SET processed = 1 WHERE id =
            {$transaction['id']} LIMIT 1");
            $stmt->execute();
        }
    }
}
```

Como puede ver, `TransactionProcessorWorker` fue bastante fácil de implementar, ya que solo teníamos que especificar el límite de memoria y preocuparnos por las acciones reales que debía realizar. No se necesita ningún manejo de errores en el `TransactionProcessorWorker` porque eso se maneja en el `AbstractWorker`.

Nota IMPORTANTE

Cuando se hereda de una clase abstracta, todos los métodos marcados como abstractos en la declaración de la clase del padre deben ser definidos por el hijo (o el propio niño también debe estar marcado como abstracto); Además, estos métodos deben definirse con la misma visibilidad (o una menos restringida). Por ejemplo, si el método abstracto se define como protegido, la implementación de la función debe definirse como protegida o pública, pero no privada.

Tomado de la [documentación de PHP para la abstracción de clase](#).

Si **no** define los métodos de clases abstractas primarias dentro de la clase secundaria, se le lanzará un **Error Fatal de PHP** como el siguiente.

Error grave: la Clase X contiene 1 método abstracto y, por lo tanto, debe declararse abstracto o implementar los métodos restantes (X :: x) en

Separación de nombres y carga automática

Técnicamente, la carga automática funciona ejecutando una devolución de llamada cuando se requiere una clase de PHP pero no se encuentra. Tales devoluciones de llamada generalmente intentan cargar estas clases.

En general, la carga automática puede entenderse como el intento de cargar archivos PHP (especialmente archivos de clase PHP, donde un archivo fuente PHP está dedicado para una clase específica) desde rutas apropiadas de acuerdo con el nombre completo de la clase (FQN) cuando se necesita una clase .

Supongamos que tenemos estas clases:

Archivo de clase para application\controllers\Base :

```
<?php
namespace application\controllers { class Base {...} }
```

Archivo de clase para application\controllers\Control :

```
<?php
namespace application\controllers { class Control {...} }
```

Archivo de clase para application\models\Page :

```
<?php
namespace application\models { class Page {...} }
```

Bajo la carpeta de origen, estas clases deben colocarse en las rutas como sus FQN respectivamente:

- Carpeta de origen
 - applications
 - controllers
 - Base.php
 - Control.php
 - models
 - Page.php

Este enfoque hace posible resolver mediante programación la ruta del archivo de clase de acuerdo con el FQN, utilizando esta función:

```
function getClassPath(string $sourceFolder, string $className, string $extension = ".php") {
    return $sourceFolder . "/" . str_replace("\\\\", "/", $className) . $extension; // note that
    "/" works as a directory separator even on Windows
}
```

La función `spl_autoload_register` nos permite cargar una clase cuando sea necesario utilizando una función definida por el usuario:

```
const SOURCE_FOLDER = __DIR__ . "/src";
spl_autoload_register(function (string $className) {
    $file = getClassPath(SOURCE_FOLDER, $className);
    if (is_readable($file)) require_once $file;
});
```

Esta función se puede ampliar aún más para utilizar métodos de carga alternativos:

```
const SOURCE_FOLDERS = [__DIR__ . "/src", "/root/src"]);
spl_autoload_register(function (string $className) {
    foreach(SOURCE_FOLDERS as $folder) {
        $extensions = [
            // do we have src/Foo/Bar.php5_int64?
            ".php" . PHP_MAJOR_VERSION . "_int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.php7?
            ".php" . PHP_MAJOR_VERSION,
            // do we have src/Foo/Bar.php_int64?
            ".php" . "int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.php5?
            ".phps",
            // do we have src/Foo/Bar.php?
            ".php"
        ];
        foreach($extensions as $ext) {
            $path = getClassPath($folder, $className, $extension);
            if(is_readable($path)) return $path;
        }
    }
});
```

Tenga en cuenta que PHP no intenta cargar las clases siempre que se carga un archivo que utiliza esta clase. Puede cargarse en medio de un script, o incluso en funciones de apagado. Esta es una de las razones por las que los desarrolladores, especialmente aquellos que usan la carga automática, deben evitar reemplazar los archivos de origen en ejecución, especialmente en archivos phar.

Vinculación dinámica

El enlace dinámico, también conocido como **invalidación de método**, es un ejemplo de **polimorfismo de tiempo de ejecución** que se produce cuando varias clases contienen implementaciones diferentes del mismo método, pero el objeto sobre el que se llamará el método es *desconocido* hasta el **tiempo de ejecución**.

Esto es útil si una determinada condición dicta qué clase se utilizará para realizar una acción, donde la acción se denomina igual en ambas clases.

```
interface Animal {
    public function makeNoise();
}
```

```

class Cat implements Animal {
    public function makeNoise
    {
        $this->meow();
    }
    ...
}

class Dog implements Animal {
    public function makeNoise {
        $this->bark();
    }
    ...
}

class Person {
    const CAT = 'cat';
    const DOG = 'dog';

    private $petPreference;
    private $pet;

    public function isCatLover(): bool {
        return $this->petPreference == self::CAT;
    }

    public function isDogLover(): bool {
        return $this->petPreference == self::DOG;
    }

    public function setPet(Animal $pet) {
        $this->pet = $pet;
    }

    public function getPet(): Animal {
        return $this->pet;
    }
}

if($person->isCatLover()) {
    $person->setPet(new Cat());
} else if($person->isDogLover()) {
    $person->setPet(new Dog());
}

$person->getPet()->makeNoise();

```

En el ejemplo anterior, la clase `Animal` (`Dog|Cat`) que creará `makeNoise` se desconoce hasta el tiempo de ejecución, según la propiedad dentro de la clase `User`.

Método y visibilidad de la propiedad

Hay tres tipos de visibilidad que puede aplicar a los métodos (*funciones de clase / objeto*) y propiedades (*variables de clase / objeto*) dentro de una clase, que proporcionan control de acceso para el método o la propiedad a la que se aplican.

Puede leer extensamente sobre esto en la [Documentación de PHP para Visibilidad OOP](#).

Público

La declaración de un método o una propiedad como `public` permite que se acceda al método o la propiedad mediante:

- La clase que lo declaró.
- Las clases que extienden la clase declarada.
- Cualquier objeto externo, clases o código fuera de la jerarquía de clases.

Un ejemplo de este acceso `public` sería:

```
class MyClass {  
    // Property  
    public $myProperty = 'test';  
  
    // Method  
    public function myMethod() {  
        return $this->myProperty;  
    }  
}  
  
$obj = new MyClass();  
echo $obj->myMethod();  
// Out: test  
  
echo $obj->myProperty;  
// Out: test
```

Protegido

La declaración de un método o una propiedad como `protected` permite que se pueda acceder al método o la propiedad mediante:

- La clase que lo declaró.
- Las clases que extienden la clase declarada.

Esto **no permite que** los objetos, clases o códigos externos fuera de la jerarquía de clases accedan a estos métodos o propiedades. Si algo que utiliza este método / propiedad no tiene acceso a él, no estará disponible y se generará un error. **Sólo las** instancias del yo declarado (o subclases del mismo) tienen acceso a él.

Un ejemplo de este acceso `protected` sería:

```
class MyClass {  
    protected $myProperty = 'test';  
  
    protected function myMethod() {  
        return $this->myProperty;  
    }  
}
```

```

}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myMethod();
    }
}

$obj = new MySubClass();
$obj->run(); // This will call MyClass::myMethod();
// Out: test

$obj->myMethod(); // This will fail.
// Out: Fatal error: Call to protected method MyClass::myMethod() from context ''

```

El ejemplo anterior señala que solo puede acceder a los elementos `protected` dentro de su propio ámbito. Esencialmente: "Lo que hay en la casa solo se puede acceder desde dentro de la casa".

Privado

La declaración de un método o una propiedad como `private` permite que se acceda al método o la propiedad mediante:

- La clase que lo declaró **Only** (no subclases).

Un método o propiedad `private` solo es visible y accesible dentro de la clase que lo creó.

Tenga en cuenta que los objetos del mismo tipo tendrán acceso a los demás miembros privados y protegidos, aunque no sean las mismas instancias.

```

class MyClass {
    private $myProperty = 'test';

    private function myPrivateMethod() {
        return $this->myProperty;
    }

    public function myPublicMethod() {
        return $this->myPrivateMethod();
    }

    public function modifyPrivatePropertyOf(MyClass $anotherInstance) {
        $anotherInstance->myProperty = "new value";
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myPublicMethod();
    }

    public function runWithPrivate() {
        echo $this->myPrivateMethod();
    }
}

```

```

$obj = new MySubClass();
$newObj = new MySubClass();

// This will call MyClass::myPublicMethod(), which will then call
// MyClass::myPrivateMethod();
$obj->run();
// Out: test

$obj->modifyPrivatePropertyOf($newObj);

$newObj->run();
// Out: new value

echo $obj->myPrivateMethod(); // This will fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context ''
echo $obj->runWithPrivate(); // This will also fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context
'MySubClass'

```

Como se indicó, solo puede acceder al método / propiedad `private` desde su clase definida.

Llamar a un constructor padre al crear una instancia de un hijo

Un error común de las clases secundarias es que, si su padre y su hijo contienen un método constructor (`__construct()`), **solo se ejecutará el constructor de la clase secundaria**. Puede haber ocasiones en las que necesite ejecutar el `__construct()` padre `__construct()` desde su hijo. Si necesita hacer eso, entonces deberá usar el resolutor `parent::__construct()`:

```
parent::__construct();
```

Ahora aprovechar que en una situación del mundo real se vería algo así como:

```

class Foo {

    function __construct($args) {
        echo 'parent';
    }
}

class Bar extends Foo {

    function __construct($args) {
        parent::__construct($args);
    }
}

```

Lo anterior ejecutará el elemento principal `__construct()` y el `echo` se ejecutará.

Palabra clave final

Def: **Final** Keyword evita que las clases secundarias invaliden un método prefijando la definición

con final. Si la clase en sí se está definiendo como final, entonces no se puede extender

Método final

```
class BaseClass {  
    public function test() {  
        echo "BaseClass::test() called\n";  
    }  
  
    final public function moreTesting() {  
        echo "BaseClass::moreTesting() called\n";  
    }  
}  
  
class ChildClass extends BaseClass {  
    public function moreTesting() {  
        echo "ChildClass::moreTesting() called\n";  
    }  
}  
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
```

Clase final

```
final class BaseClass {  
    public function test() {  
        echo "BaseClass::test() called\n";  
    }  
  
    // Here it doesn't matter if you specify the function as final or not  
    final public function moreTesting() {  
        echo "BaseClass::moreTesting() called\n";  
    }  
}  
  
class ChildClass extends BaseClass {  
}  
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)
```

Constantes finales: a diferencia de Java, la palabra clave `final` no se usa para las constantes de clase en PHP. Utilice la palabra clave `const` lugar.

¿Por qué tengo que usar `final`?

1. La prevención de la cadena de herencia masiva de la fatalidad
2. Composición estimulante
3. Forzar al desarrollador a pensar en la API pública del usuario
4. Forzar al desarrollador a reducir la API pública de un objeto
5. Una clase `final` siempre se puede hacer extensible.
6. `extends` roturas de encapsulación.
7. No necesitas esa flexibilidad.
8. Eres libre de cambiar el código

Cuándo evitar `final`: las clases finales solo funcionan de manera efectiva bajo los siguientes supuestos:

1. Hay una abstracción (interfaz) que implementa la clase final
2. Toda la API pública de la clase final es parte de esa interfaz

\$ esto, auto y estático más el singleton

Use `$this` para referirse al objeto actual. Use `self` para referirse a la clase actual. En otras palabras, use `$this->member` para `$this->member` no estáticos, use `self::$member` para miembros estáticos.

En el siguiente ejemplo, `sayHello()` y `sayGoodbye()` están usando `self` y `$this` diferencia se puede observar aquí.

```
class Person {  
    private $name;  
  
    public function __construct($name) {  
        $this->name = $name;  
    }  
  
    public function getName() {  
        return $this->name;  
    }  
  
    public function getTitle() {  
        return $this->getName()." the person";  
    }  
  
    public function sayHello() {  
        echo "Hello, I'm ".$this->getTitle()."<br/>";  
    }  
  
    public function sayGoodbye() {  
        echo "Goodbye from ".self::$title."<br/>";  
    }  
}  
  
class Geek extends Person {  
    public function __construct($name) {  
        parent::__construct($name);  
    }  
  
    public function getTitle() {  
        return $this->getName()." the geek";  
    }  
}  
  
$geekObj = new Geek("Ludwig");  
$geekObj->sayHello();  
$geekObj->sayGoodbye();
```

`static` refiere a cualquier clase en la jerarquía en la que llamas al método. Permite una mejor reutilización de las propiedades de clase estáticas cuando las clases se heredan.

Considere el siguiente código:

```

class Car {
    protected static $brand = 'unknown';

    public static function brand() {
        return self::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();

```

Esto no produce el resultado que desea:

desconocido
desconocido
desconocido

Esto se debe a que `self` refiere a la clase `Car` cuando se llama a la `brand()` método `brand()`.

Para referirse a la clase correcta, necesita usar `static` lugar:

```

class Car {
    protected static $brand = 'unknown';

    public static function brand() {
        return static::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();

```

Esto produce el resultado deseado:

desconocido
BMW
Mercedes

Véase también [Enlace estático tardío](#)

El singleton

Si tiene un objeto que es costoso crear o representa una conexión a algún recurso externo que desea reutilizar, es decir, una conexión de base de datos donde no existe una agrupación de conexiones o un socket para algún otro sistema, puede usar las palabras clave `static` y `self` en un clase para hacer un singleton. Hay opiniones fuertes acerca de si el patrón de singleton debe o no debe usarse, pero tiene sus usos.

```
class Singleton {  
    private static $instance = null;  
  
    public static function getInstance() {  
        if(!isset(self::$instance)){  
            self::$instance = new self();  
        }  
  
        return self::$instance;  
    }  
  
    private function __construct() {  
        // Do constructor stuff  
    }  
}
```

Como puede ver en el código de ejemplo, estamos definiendo una `$instance` privada de propiedad estática `$instance` para contener la referencia del objeto. Como esto es estático, esta referencia se comparte entre TODOS los objetos de este tipo.

El método `getInstance()` utiliza un método conocido como creación de instancias perezosa para retrasar la creación del objeto hasta el último momento posible, ya que no desea tener objetos no utilizados en la memoria que nunca se pretendió usar. También ahorra tiempo y CPU en la carga de la página, no tiene que cargar más objetos de los necesarios. El método comprueba si el objeto está establecido, lo crea, si no, y lo devuelve. Esto asegura que solo un objeto de este tipo sea creado.

También estamos configurando el constructor para que sea privado para garantizar que nadie lo cree con la `new` palabra clave desde el exterior. Si necesita heredar de esta clase, simplemente cambie las palabras clave `private` a `protected`.

Para usar este objeto solo escribe lo siguiente:

```
$singleton = Singleton::getInstance();
```

Ahora le imploro que use la inyección de dependencia donde pueda y apunte a objetos acoplados de forma flexible, pero a veces eso no es razonable y el patrón de singleton puede ser de utilidad.

Autocarga

Nadie quiere `require` o `include` cada vez que se usa una clase o herencia. Debido a que puede ser doloroso y es fácil de olvidar, PHP está ofreciendo el llamado autoloading. Si ya está utilizando Composer, lea acerca de la [carga automática utilizando Composer](#).

¿Qué es exactamente la carga automática?

El nombre básicamente lo dice todo. No tiene que obtener el archivo donde se almacena la clase solicitada, pero PHP lo *carga automáticamente*.

¿Cómo puedo hacer esto en PHP básico sin código de terceros?

Existe la función `__autoload`, pero se considera una mejor práctica usar `spl_autoload_register`. PHP considerará estas funciones cada vez que no se defina una clase dentro del espacio dado. Entonces, agregar la carga automática a un proyecto existente no es un problema, ya que las clases definidas (a través de `require` ie) funcionarán como antes. En aras de la precisión, los siguientes ejemplos usarán funciones anónimas, si usa PHP <5.3, puede definir la función y pasar su nombre como argumento a `spl_autoload_register`.

Ejemplos

```
spl_autoload_register(function ($className) {
    $path = sprintf('%s.php', $className);
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

El código anterior simplemente intenta incluir un nombre de archivo con el nombre de la clase y la extensión anexada ".php" usando `sprintf`. Si `FooBar` necesita ser cargado, parece que `FooBar.php` existe y si es así lo incluye.

Por supuesto, esto puede extenderse para adaptarse a las necesidades individuales del proyecto. Si _ dentro de un nombre de clase se usa para agrupar, por ejemplo, `User_Post` y `User_Image` refieren a `User`, ambas clases se pueden mantener en una carpeta llamada "User" como:

```
spl_autoload_register(function ($className) {
    // replace _ by / or \ (depending on OS)
    $path = sprintf('%s.php', str_replace('_', DIRECTORY_SEPARATOR, $className));
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

La clase `User_Post` ahora se cargará desde "User / Post.php", etc.

`spl_autoload_register` puede adaptarse a diversas necesidades. Todos sus archivos con clases se llaman "class.CLASSNAME.php"? No hay problema. Varios anidamientos (`User_Post_Content => "User / Post / Content.php"`)? No hay problema tampoco.

Si desea un mecanismo de carga automática más elaborado, y aún no desea incluir Composer, puede trabajar sin agregar bibliotecas de terceros.

```
spl_autoload_register(function ($className) {
    $path = sprintf('%1$s%2$s%3$s.php',
        // %1$s: get absolute path
        realpath(dirname(__FILE__)),
        // %2$s: / or \ (depending on OS)
        DIRECTORY_SEPARATOR,
        // %3$s: don't worry about caps or not when creating the files
        strtolower(
            // replace _ by / or \ (depending on OS)
            str_replace('_', DIRECTORY_SEPARATOR, $className)
        )
    );
    if (file_exists($path)) {
        include $path;
    } else {
        throw new Exception(
            sprintf('Class with name %1$s not found. Looked in %2$s.',
                $className,
                $path
            )
        );
    }
});
```

Usando autocargadores como este, felizmente puede escribir código como este:

```
require_once './autoload.php'; // where spl_autoload_register is defined

$foo = new Foo_Bar(new Hello_World());
```

Utilizando clases:

```
class Foo_Bar extends Foo {}

class Hello_World implements Demo_Classes {}
```

Estos ejemplos incluirán clases de `foo/bar.php`, `foo.php`, `hello/world.php` y `demo/classes.php`.

Clases anónimas

Se introdujeron clases anónimas en PHP 7 para permitir la creación fácil de objetos únicos y rápidos. Pueden tomar argumentos de constructor, extender otras clases, implementar interfaces y usar rasgos al igual que las clases normales.

En su forma más básica, una clase anónima se parece a lo siguiente:

```
new class("constructor argument") {
    public function __construct($param) {
        var_dump($param);
```

```
    }
}; // string(20) "constructor argument"
```

Anidar una clase anónima dentro de otra clase no le da acceso a métodos o propiedades privadas o protegidas de esa clase externa. El acceso a los métodos protegidos y las propiedades de la clase externa se puede obtener extendiendo la clase externa de la clase anónima. El acceso a las propiedades privadas de la clase externa se puede obtener pasándolas al constructor de la clase anónima.

Por ejemplo:

```
class Outer {
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1() {
        return 3;
    }

    public function func2() {
        // passing through the private $this->prop property
        return new class($this->prop) extends Outer {
            private $prop3;

            public function __construct($prop) {
                $this->prop3 = $prop;
            }

            public function func3() {
                // accessing the protected property Outer::$prop2
                // accessing the protected method Outer::func1()
                // accessing the local property self::$prop3 that was private from
Outer::$prop
                return $this->prop2 + $this->func1() + $this->prop3;
            }
        };
    }
}

echo (new Outer)->func2()->func3(); // 6
```

Definiendo una clase básica

Un objeto en PHP contiene variables y funciones. Los objetos normalmente pertenecen a una clase, que define las variables y funciones que contendrán todos los objetos de esta clase.

La sintaxis para definir una clase es:

```
class Shape {
    public $sides = 0;

    public function description() {
        return "A shape with $this->sides sides.";
    }
}
```

Una vez que se define una clase, puede crear una instancia usando:

```
$myShape = new Shape();
```

A las variables y funciones en el objeto se accede de esta manera:

```
$myShape = new Shape();
$myShape->sides = 6;

print $myShape->description(); // "A shape with 6 sides"
```

Constructor

Las clases pueden definir un `__construct()` especial `__construct()`, que se ejecuta como parte de la creación del objeto. Esto se usa a menudo para especificar los valores iniciales de un objeto:

```
class Shape {
    public $sides = 0;

    public function __construct($sides) {
        $this->sides = $sides;
    }

    public function description() {
        return "A shape with $this->sides sides.";
    }
}

$myShape = new Shape(6);

print $myShape->description(); // A shape with 6 sides
```

Extendiendo otra clase

Las definiciones de clase pueden extender las definiciones de clase existentes, agregar nuevas variables y funciones, así como modificar aquellas definidas en la clase principal.

Aquí hay una clase que amplía el ejemplo anterior:

```
class Square extends Shape {
    public $sideLength = 0;

    public function __construct($sideLength) {
        parent::__construct(4);

        $this->sideLength = $sideLength;
    }

    public function perimeter() {
        return $this->sides * $this->sideLength;
    }
}
```

```
public function area() {
    return $this->sideLength * $this->sideLength;
}
}
```

La clase `Square` contiene variables y comportamiento tanto para la clase `Shape` como para la clase `Square`:

```
$mySquare = new Square(10);

print $mySquare->description() // A shape with 4 sides

print $mySquare->perimeter() // 40

print $mySquare->area() // 100
```

Lea Clases y objetos en línea: <https://riptutorial.com/es/php/topic/504/clases-y-objetos>

Capítulo 18: Cliente de jabón

Sintaxis

- `__getFunctions ()` // Devuelve la matriz de funciones para el servicio (solo en modo WSDL)
- `__getTypes ()` // Devuelve una matriz de tipos para el servicio (solo en modo WSDL)
- `__getLastRequest ()` // Devuelve XML de la última solicitud (requiere la opción de `trace`)
- `__getLastRequestHeaders ()` // Devuelve los encabezados de la última solicitud (requiere la opción de `trace`)
- `__getLastResponse ()` // Devuelve XML de la última respuesta (requiere la opción de `trace`)
- `__getLastResponseHeaders ()` // Devuelve los encabezados de la última respuesta (requiere la opción de `trace`)

Parámetros

Parámetro	Detalles
\$ wsdl	URI de WSDL o <code>NULL</code> si se utiliza el modo no WSDL
\$ opciones	Array de opciones para <code>SoapClient</code> . El modo no WSDL requiere <code>location</code> y <code>uri</code> para establecer, todas las demás opciones son opcionales. Consulte la tabla a continuación para ver los posibles valores.

Observaciones

La clase `SoapClient` está equipada con un método `__call`. Esto *no* debe ser llamado directamente. En su lugar, esto le permite hacer:

```
$soap->requestInfo(['a', 'b', 'c']);
```

Esto llamará al método de `requestInfo` SOAP.

Tabla de posibles valores de `$options` (*Array de pares clave / valor*):

Opción	Detalles
ubicación	URL del servidor SOAP. <i>Requerido</i> en modo no WSDL. Se puede utilizar en modo WSDL para anular la URL.
uri	Espacio de nombres de destino del servicio SOAP. <i>Requerido</i> en modo no WSDL.
estilo	Los valores posibles son <code>SOAP_RPC</code> o <code>SOAP_DOCUMENT</code> . Sólo válido en modo

Opción	Detalles
	no WSDL.
utilizar	Los valores posibles son <code>SOAP_ENCODED</code> o <code>SOAP_LITERAL</code> . Sólo válido en modo no WSDL.
soap_version	Los valores posibles son <code>SOAP_1_1</code> (<i>predeterminado</i>) o <code>SOAP_1_2</code> .
autenticación	Habilitar la autenticación HTTP. Los valores posibles son <code>SOAP_AUTHENTICATION_BASIC</code> (<i>predeterminado</i>) o <code>SOAP_AUTHENTICATION_DIGEST</code> .
iniciar sesión	Nombre de usuario para autenticación HTTP
contraseña	Contraseña para la autenticación HTTP
proxy_host	URL del servidor proxy
Puerto proxy	Puerto de servidor proxy
proxy_login	Nombre de usuario para proxy
proxy_password	Contraseña para proxy
local_cert	Ruta al certificado de cliente HTTPS (para autenticación)
frase de contraseña	Frase de contraseña para el certificado de cliente HTTPS
compresión	Comprimir la solicitud / respuesta. El valor es una máscara de bits de <code>SOAP_COMPRESSION_ACCEPT</code> con <code>SOAP_COMPRESSION_GZIP</code> o <code>SOAP_COMPRESSION_DEFLATE</code> . Por ejemplo: <code>SOAP_COMPRESSION_ACCEPT \SOAP_COMPRESSION_GZIP</code> .
codificación	Codificación interna de caracteres (TODO: valores posibles)
rastro	<code>Booleano</code> , por defecto es <code>FALSE</code> . Habilita el seguimiento de las solicitudes para que las fallas se puedan retroceder. Habilita el uso de <code>__getLastRequest()</code> , <code>__getLastRequestHeaders()</code> , <code>__getLastResponse()</code> y <code>__getLastResponseHeaders()</code> .
mapa de clase	Mapear los tipos WSDL a las clases de PHP. El valor debe ser una matriz con tipos WSDL como claves y nombres de clase de PHP como valores.
excepciones	Valor <code>booleano</code> . En caso de excepciones de errores SOAP (de tipo 'SoapFault).
el tiempo de conexión expiro	Tiempo de espera (en segundos) para la conexión al servicio SOAP.

Opción	Detalles
mapa de tipo	Matriz de asignaciones de tipo. La matriz debe ser pares clave / valor con las siguientes claves: <code>type_name</code> , <code>type_ns</code> (URI del espacio de nombres), <code>from_xml</code> (devolución de llamada que acepta un parámetro de cadena) y <code>to_xml</code> (devolución de llamada que acepta un parámetro de objeto).
cache_wsdl	Cómo (si lo hace) el archivo WSDL debe ser almacenado en caché. Los valores posibles son <code>WSDL_CACHE_NONE</code> , <code>WSDL_CACHE_DISK</code> , <code>WSDL_CACHE_MEMORY</code> o <code>WSDL_CACHE_BOTH</code> .
agente de usuario	Cadena para usar en el encabezado <code>User-Agent</code> .
stream_context	Un recurso para un contexto.
características	Máscara de bits de <code>SOAP_SINGLE_ELEMENT_ARRAYS</code> , <code>SOAP_USE_XSI_ARRAY_TYPE</code> , <code>SOAP_WAIT_ONE_WAY_CALLS</code> .
mantener viva	(Versión de PHP>= 5.4 solamente) Valor booleano . Envíe el encabezado <code>Connection: Keep-Alive</code> (<code>TRUE</code>) o <code>Connection: Close</code> header (<code>FALSE</code>).
ssl_method	(Versión de PHP>= 5.5 solamente) Qué versión de SSL / TLS usar. Los valores posibles son <code>SOAP_SSL_METHOD_TLS</code> , <code>SOAP_SSL_METHOD_SSLv2</code> , <code>SOAP_SSL_METHOD_SSLv3</code> o <code>SOAP_SSL_METHOD_SSLv23</code> .

Problema con PHP de 32 bits : en PHP de 32 bits, las cadenas numéricas mayores de 32 bits que se convierten automáticamente en enteros por `xs:long` resultarán en que alcance el límite de 32 bits, `2147483647` en `2147483647` . Para `__soapCall()` esto, lance las cadenas para que floten antes de pasarlo a `__soapCall()` .

Examples

Modo WSDL

Primero, cree un nuevo objeto `SoapClient` , pasando la URL al archivo WSDL y, opcionalmente, una variedad de opciones.

```
// Create a new client object using a WSDL URL
$soap = new SoapClient('https://example.com/soap.wsdl', [
    # This array and its values are optional
    'soap_version' => SOAP_1_2,
    'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
    'cache_wsdl' => WSDL_CACHE_BOTH,
    # Helps with debugging
    'trace' => TRUE,
    'exceptions' => TRUE
]);
```

Luego usa el objeto `$soap` para llamar a tus métodos SOAP.

```
$result = $soap->requestData(['a', 'b', 'c']);
```

Modo no WSDL

Esto es similar al modo WSDL, excepto que pasamos `NULL` como el archivo WSDL y nos aseguramos de establecer la `location` y las opciones de `uri`.

```
$soap = new SoapClient(NULL, [
    'location' => 'https://example.com/soap/endpoint',
    'uri' => 'namespace'
]);
```

Mapas de clase

Al crear un cliente SOAP en PHP, también puede establecer una clave de `classmap` en la matriz de configuración. Este `classmap` define qué tipos definidos en el WSDL deben asignarse a clases reales, en lugar del `StdClass` predeterminado. La razón por la que querría hacer esto es porque puede completar automáticamente los campos y las llamadas a métodos en estas clases, en lugar de tener que adivinar qué campos se configuran en la `StdClass` regular.

```
class MyAddress {
    public $country;
    public $city;
    public $full_name;
    public $postal_code; // or zip_code
    public $house_number;
}

class MyBook {
    public $name;
    public $author;

    // The classmap also allows us to add useful functions to the objects
    // that are returned from the SOAP operations.
    public function getShortDescription() {
        return "{$this->name}, written by {$this->author}";
    }
}

$soap_client = new SoapClient($link_to_wsdl, [
    // Other parameters
    "classmap" => [
        "Address" => MyAddress::class, // ::class simple returns class as string
        "Book" => MyBook::class,
    ]
]);
```

Después de configurar el mapa de clase, siempre que realice una determinada operación que devuelva un tipo de `Address` o `Book`, `SoapClient` creará una instancia de esa clase, llenará los campos con los datos y los devolverá desde la llamada de la operación.

```

// Lets assume 'getAddress(1234)' returns an Address by ID in the database
$address = $soap_client->getAddress(1234);

// $address is now of type MyAddress due to the classmap
echo $address->country;

// Lets assume the same for 'getBook(1234)'
$book = $soap_client->getBook(124);

// We can not use other functions defined on the MyBook class
echo $book->getShortDescription();

// Any type defined in the WSDL that is not defined in the classmap
// will become a regular StdClass object
$author = $soap_client->getAuthor(1234);

// No classmap for Author type, $author is regular StdClass.
// We can still access fields, but no auto-completion and no custom functions
// to define for the objects.
echo $author->name;

```

Rastreo de solicitud y respuesta SOAP

A veces queremos ver lo que se envía y recibe en la solicitud de SOAP. Los siguientes métodos devolverán el XML en la solicitud y respuesta:

```

SoapClient::__getLastRequest()
SoapClient::__getLastRequestHeaders()
SoapClient::__getLastResponse()
SoapClient::__getLastResponseHeaders()

```

Por ejemplo, supongamos que tenemos una constante de `ENVIRONMENT` y cuando el valor de esta constante se establece en `DEVELOPMENT`, queremos repetir toda la información cuando la llamada a `getAddress` un error. Una solución podría ser:

```

try {
    $address = $soap_client->getAddress(1234);
} catch (SoapFault $e) {
    if (ENVIRONMENT === 'DEVELOPMENT') {
        var_dump(
            $soap_client->__getLastRequestHeaders(),
            $soap_client->__getLastRequest(),
            $soap_client->__getLastResponseHeaders(),
            $soap_client->__getLastResponse()
        );
    }
    ...
}

```

Lea Cliente de jabón en línea: <https://riptutorial.com/es/php/topic/633/cliente-de-jabon>

Capítulo 19: Comentarios

Observaciones

Tenga en cuenta los siguientes consejos cuando decida cómo comentar su código:

- Siempre debe escribir su código como si los comentarios no existieran, utilizando nombres de funciones y variables bien elegidos.
- Los comentarios están destinados a comunicarse con otros seres humanos, no para repetir lo que está escrito en el código.
- Existen varias guías de estilo de comentarios php (por ejemplo, [pear](#) , [zend](#) , etc). ¡Averigua cuál usa tu compañía y úsala de manera consistente!

Examples

Comentarios de una sola línea

El comentario de una sola línea comienza con "://" o "#". Cuando se encuentre, todo el texto de la derecha será ignorado por el intérprete de PHP.

```
// This is a comment  
  
# This is also a comment  
  
echo "Hello World!"; // This is also a comment, beginning where we see "//"
```

Comentarios multilínea

El comentario multilínea se puede usar para comentar grandes bloques de código. Comienza con /* y termina con */ .

```
/* This is a multi-line comment.  
It spans multiple lines.  
This is still part of the comment.  
*/
```

Lea Comentarios en línea: <https://riptutorial.com/es/php/topic/6852/comentarios>

Capítulo 20: Cómo desglosar una URL

Introducción

A medida que codifiques PHP, lo más probable es que te pongas en una posición en la que necesites dividir una URL en varias partes. Obviamente, hay más de una forma de hacerlo dependiendo de sus necesidades. Este artículo explicará esas formas para que usted pueda encontrar lo que funciona mejor para usted.

Examples

Usando parse_url ()

`parse_url ()`: esta función analiza una URL y devuelve una matriz asociativa que contiene cualquiera de los diversos componentes de la URL que están presentes.

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
)
```

Si necesitas separar el camino puedes usar Explode

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');
$url['sections'] = explode('/', $url['path']);

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
    [sections] => Array
        (
            [0] =>
            [1] => project
            [2] => controller
            [3] => action
            [4] => param1
            [5] => param2
        )
)
```

Si necesita la última parte de la sección, puede usar `end ()` de la siguiente manera:

```
$last = end($url['sections']);
```

Si la URL contiene GET vars, también puede recuperarlos.

```
$url = parse_url('http://example.com?var1=value1&var2=value2');

Array
(
    [scheme] => http
    [host] => example.com
    [query] => var1=value1&var2=value2
)
```

Si desea desglosar las variables de consulta, puede usar `parse_str()` de la siguiente manera:

```
$url = parse_url('http://example.com?var1=value1&var2=value2');
parse_str($url['query'], $parts);

Array
(
    [var1] => value1
    [var2] => value2
)
```

Utilizando explotar ()

`explode()`: devuelve una matriz de cadenas, cada una de las cuales es una subcadena de cadena formada dividiéndola en los límites formados por el delimitador de cadena.

Esta función es bastante sencilla.

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = explode('/', $url);

Array
(
    [0] => http:
    [1] =>
    [2] => example.com
    [3] => project
    [4] => controller
    [5] => action
    [6] => param1
    [7] => param2
)
```

Puedes recuperar la última parte de la URL haciendo esto:

```
$last = end($parts);
// Output: param2
```

También puede navegar dentro de la matriz utilizando `sizeof()` en combinación con un operador matemático como este:

```
echo $parts[sizeof($parts)-2];
// Output: param1
```

Usando basename ()

basename (): dada una cadena que contiene la ruta a un archivo o directorio, esta función devolverá el componente de nombre final.

Esta función devolverá solo la última parte de una URL

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = basename($url);
// Output: param2
```

Si su URL tiene más cosas y lo que necesita es el nombre de directorio que contiene el archivo, puede usarlo con dirname () de la siguiente manera:

```
$url = "http://example.com/project/controller/action/param1/param2/index.php";
$parts = basename(dirname($url));
// Output: param2
```

Lea Cómo desglosar una URL en línea: <https://riptutorial.com/es/php/topic/10847/como-desglosar-una-url>

Capítulo 21: Cómo detectar la dirección IP del cliente

Examples

Uso correcto de `HTTP_X_FORWARDED_FOR`

A la luz de las últimas vulnerabilidades de [httpoxy](#), hay otra variable, que es ampliamente mal utilizada.

`HTTP_X_FORWARDED_FOR` se usa a menudo para detectar la dirección IP del cliente, pero sin ninguna verificación adicional, esto puede llevar a problemas de seguridad, especialmente cuando esta IP se usa más adelante para la autenticación o en consultas SQL sin saneamiento.

La mayoría de los ejemplos de código disponibles ignoran el hecho de que `HTTP_X_FORWARDED_FOR` puede considerarse realmente como información proporcionada por el propio cliente y, por lo tanto, **no es** una fuente confiable para detectar la dirección IP de los clientes. Algunas de las muestras agregan una advertencia sobre el posible uso indebido, pero aún no tienen ninguna verificación adicional en el propio código.

Así que aquí hay un ejemplo de la función escrita en PHP, cómo detectar la dirección IP de un cliente, si sabe que ese cliente puede estar detrás de un proxy y sabe que se puede confiar en este proxy. Si no conoces ningún proxy de confianza, puedes usar `REMOTE_ADDR`

```
function get_client_ip()
{
    // Nothing to do without any reliable information
    if (!isset($_SERVER['REMOTE_ADDR'])) {
        return NULL;
    }

    // Header that is used by the trusted proxy to refer to
    // the original IP
    $proxy_header = "HTTP_X_FORWARDED_FOR";

    // List of all the proxies that are known to handle 'proxy_header'
    // in known, safe manner
    $trusted_proxies = array("2001:db8::1", "192.168.50.1");

    if (in_array($_SERVER['REMOTE_ADDR'], $trusted_proxies)) {

        // Get IP of the client behind trusted proxy
        if (array_key_exists($proxy_header, $_SERVER)) {

            // Header can contain multiple IP-s of proxies that are passed through.
            // Only the IP added by the last proxy (last IP in the list) can be trusted.
            $client_ip = trim(end(explode(",", $_SERVER[$proxy_header])));

            // Validate just in case
            if (filter_var($client_ip, FILTER_VALIDATE_IP)) {
                return $client_ip;
            }
        }
    }
}
```

```
    } else {
        // Validation failed - beat the guy who configured the proxy or
        // the guy who created the trusted proxy list?
        // TODO: some error handling to notify about the need of punishment
    }
}

// In all other cases, REMOTE_ADDR is the ONLY IP we can trust.
return $_SERVER['REMOTE_ADDR'];
}

print get_client_ip();
```

Lea Cómo detectar la dirección IP del cliente en línea:

<https://riptutorial.com/es/php/topic/5058/como-detectar-la-direccion-ip-del-cliente>

Capítulo 22: Compilar extensiones de PHP

Examples

Compilando en linux

Para compilar una extensión de PHP en un entorno Linux típico, hay algunos requisitos previos:

- Habilidades básicas de Unix (poder operar "make" y un compilador de C)
- Un compilador ANSI C
- El código fuente de la extensión PHP que desea compilar.

Generalmente hay dos formas de compilar una extensión de PHP. Puede compilar **estáticamente** la extensión en el binario de PHP, o compilarla como un módulo **compartido** cargado por su binario de PHP en el inicio. Los módulos compartidos son más probables ya que le permiten agregar o eliminar extensiones sin reconstruir todo el binario de PHP. Este ejemplo se centra en la opción compartida.

Si instaló PHP a través de su administrador de paquetes (`apt-get install`, `yum install`, etc.) necesitará instalar el paquete `-dev` para PHP, que incluirá los archivos de encabezado PHP y el script `phpize` necesarios para que funcione el entorno de compilación. El paquete podría llamarse algo así como `php5-dev` o `php7-dev`, pero asegúrese de usar su administrador de paquetes para buscar el nombre apropiado usando los repositorios de su distro. Pueden diferir.

Si compiló PHP desde la fuente, lo más probable es que los archivos de encabezado ya existan en su sistema (generalmente en `/usr/include` o `/usr/local/include`).

Pasos para compilar

Después de verificar para asegurarse de que tiene todos los requisitos previos necesarios para compilar, puede dirigirse a pecl.php.net, seleccionar una extensión que desee compilar y descargar la bola de alquitrán.

1. Desembale la bola de alquitrán (por ejemplo, `tar xfvz yaml-2.0.0RC8.tgz`)
2. Ingrese el directorio donde se desempaquetó el archivo y ejecute `phpize`
3. Ahora debería ver una `.configure` comando. `.configure` recién creada si todo salió bien, ejecute ese `./configure`
4. Ahora necesitarás ejecutar `make`, que compilará la extensión.
5. Finalmente, `make install` copiará el archivo binario de extensión compilado a su directorio de extensión.

El `make install` paso normalmente proporcionará la ruta de instalación para usted que se ha copiado la extensión. Esto suele estar en `/usr/lib/`, por ejemplo, podría ser algo como `/usr/lib/php5/20131226/yaml.so`. Pero esto depende de su configuración de PHP (es decir, `--with-prefix`) y la versión específica de la API. El número de API se incluye en la ruta para mantener

las extensiones creadas para diferentes versiones de API en ubicaciones separadas.

Cargando la extensión en PHP

Para cargar la extensión en PHP, encuentre su archivo php.ini cargado para el SAPI apropiado, y agregue la `extension=yaml.so` línea `extension=yaml.so` luego reinicie PHP. Cambie `yaml.so` al nombre de la extensión real que instaló, por supuesto.

Para una extensión Zend, debe proporcionar la ruta completa al archivo de objeto compartido. Sin embargo, para las extensiones PHP normales, esta ruta se deriva de la directiva `extension_dir` en su configuración cargada, o del entorno `$PATH` durante la configuración inicial.

Lea [Compilar extensiones de PHP en línea](https://riptutorial.com/es/php/topic/6767/compilar-extensiones-de-php): <https://riptutorial.com/es/php/topic/6767/compilar-extensiones-de-php>

Capítulo 23: Constantes

Sintaxis

- define (cadena \$ nombre, valor mezclado de \$ [, bool \$ case_insensitive = false])
- const CONSTANT_NAME = VALUE;

Observaciones

Las constantes se utilizan para almacenar los valores que no se deben cambiar más adelante. También se utilizan a menudo para almacenar los parámetros de configuración, especialmente aquellos que definen el entorno (desarrollo / producción).

Las constantes tienen tipos como variables pero no todos los tipos se pueden usar para inicializar una constante. Los objetos y los recursos no se pueden utilizar como valores para constantes en absoluto. Las matrices se pueden usar como constantes a partir de PHP 5.6

Algunos nombres constantes están reservados por PHP. Estos incluyen `true`, `false`, `null` así como muchas constantes específicas del módulo.

Las constantes son usualmente nombradas usando letras mayúsculas.

Examples

Comprobando si se define constante

Simple cheque

Para verificar si la constante está definida use la función `defined`. Tenga en cuenta que a esta función no le importa el valor de la constante, solo le importa si la constante existe o no. Incluso si el valor de la constante es `null` o `false` la función seguirá siendo `true`.

```
<?php

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined" ; // prints "GOOD is defined"

    if (GOOD) {
        print "GOOD is true" ; // does not print anything, since GOOD is false
    }
}

if (!defined("AWESOME")) {
    define("AWESOME", true); // awesome was not defined. Now we have defined it
}
```

Tenga en cuenta que la constante se vuelve "visible" en su código solo **después de la línea** donde lo ha definido:

```
<?php

if (defined("GOOD")) {
    print "GOOD is defined"; // doesn't print anything, GOOD is not defined yet.
}

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined"; // prints "GOOD is defined"
}
```

Obteniendo todas las constantes definidas

Para obtener todas las constantes definidas, incluidas las creadas por PHP, use la función `get_defined_constants`:

```
<?php

$constants = get_defined_constants();
var_dump($constants); // pretty large list
```

Para obtener solo las constantes que definió su aplicación, llame a la función al principio y al final de su script (normalmente después del proceso de arranque):

```
<?php

$constants = get_defined_constants();

define("HELLO", "hello");
define("WORLD", "world");

$new_constants = get_defined_constants();

$myconstants = array_diff_assoc($new_constants, $constants);
var_export($myconstants);

/*
Output:

array (
    'HELLO' => 'hello',
    'WORLD' => 'world',
)
*/
```

A veces es útil para la depuración.

Definiendo constantes

Las constantes se crean utilizando la sentencia `const` o la función `define`. La convención es usar letras MAYÚSCULAS para nombres constantes.

Definir constante utilizando valores explícitos.

```
const PI = 3.14; // float
define("EARTH_IS_FLAT", false); // boolean
const UNKNOWN = null; // null
define("APP_ENV", "dev"); // string
const MAX_SESSION_TIME = 60 * 60; // integer, using (scalar) expressions is ok

const APP_LANGUAGES = ["de", "en"]; // arrays

define("BETTER_APP_LANGUAGES", ["lu", "de"]); // arrays
```

Definir constante utilizando otra constante.

Si tienes una constante puedes definir otra basada en ella:

```
const TAU = PI * 2;
define("EARTH_IS_ROUND", !EARTH_IS_FLAT);
define("MORE_UNKNOWN", UNKNOWN);
define("APP_ENV_UPPERCASE", strtoupper(APP_ENV)); // string manipulation is ok too
// the above example (a function call) does not work with const:
// const TIME = time(); # fails with a fatal error! Not a constant scalar expression
define("MAX_SESSION_TIME_IN_MINUTES", MAX_SESSION_TIME / 60);

const APP_FUTURE_LANGUAGES = [-1 => "es"] + APP_LANGUAGES; // array manipulations

define("APP_BETTER_FUTURE_LANGUAGES", array_merge(["fr"], APP_BETTER_LANGUAGES));
```

Constantes reservadas

Algunos nombres constantes están reservados por PHP y no pueden ser redefinidos. Todos estos ejemplos fallarán:

```
define("true", false); // internal constant
define("false", true); // internal constant
define("CURLOPT_AUTOREFERER", "something"); // will fail if curl extension is loaded
```

Y se emitirá un Aviso:

```
Constant ... already defined in ...
```

Condicional define

Si tiene varios archivos donde puede definir la misma variable (por ejemplo, su configuración principal, su configuración local), la siguiente sintaxis puede ayudar a evitar conflictos:

```
defined("PI") || define("PI", 3.1415); // "define PI if it's not yet defined"
```

`const` VS `define`

`define` es una expresión en tiempo de ejecución, mientras que `const` un tiempo de compilación uno.

Por lo tanto, `define` permite valores dinámicos (es decir, llamadas a funciones, variables, etc.) e incluso nombres dinámicos y definiciones condicionales. Sin embargo, siempre se está definiendo en relación con el espacio de nombres raíz.

`const` es estática (como en permite solo operaciones con otras constantes, escalares o matrices, y solo un conjunto restringido de ellas, las denominadas *expresiones escalares constantes*, es decir, operadores aritméticos, lógicos y de comparación, así como la eliminación de referencias de matrices), pero son espacios de nombres automáticamente prefijado con el espacio de nombres actualmente activo.

`const` solo admite otras constantes y escalares como valores y no operaciones.

Constantes de clase

Las constantes se pueden definir dentro de las clases usando una palabra clave `const`.

```
class Foo {
    const BAR_TYPE = "bar";

    // reference from inside the class using self:::
    public function myMethod() {
        return self::BAR_TYPE;
    }
}

// reference from outside the class using <ClassName>:::
echo Foo::BAR_TYPE;
```

Esto es útil para almacenar tipos de artículos.

```
<?php

class Logger {
    const LEVEL_INFO = 1;
    const LEVEL_WARNING = 2;
    const LEVEL_ERROR = 3;
```

```

// we can even assign the constant as a default value
public function log($message, $level = self::LEVEL_INFO) {
    echo "Message level " . $level . ":" . $message;
}

$logger = new Logger();
$logger->log("Info"); // Using default value
$logger->log("Warning", $logger::LEVEL_WARNING); // Using var
$logger->log("Error", Logger::LEVEL_ERROR); // using class

```

Matrices constantes

Las matrices se pueden usar como constantes simples y constantes de clase desde la versión PHP 5.6 en adelante:

Ejemplo de clase constante

```

class Answer {
    const C = [2, 4];
}

print Answer::C[1] . Answer::C[0]; // 42

```

Ejemplo de constante llana

```

const ANSWER = [2, 4];
print ANSWER[1] . ANSWER[0]; // 42

```

También desde la versión PHP 7.0 esta funcionalidad fue portada a la función de `define` para constantes simples.

```

define('VALUES', [2, 3]);
define('MY_ARRAY', [
    1,
    VALUES,
]);

print MY_ARRAY[1][1]; // 3

```

Usando constantes

Para usar la constante simplemente usa su nombre:

```

if (EARTH_IS_FLAT) {
    print "Earth is flat";
}

print APP_ENV_UPPERCASE;

```

o si no conoce el nombre de la constante de antemano, use la función `constant`:

```
// this code is equivalent to the above code
$const1 = "EARTH_IS_FLAT";
$const2 = "APP_ENV_UPPERCASE";

if (constant($const1)) {
    print "Earth is flat";
}

print constant($const2);
```

Lea Constantes en línea: <https://riptutorial.com/es/php/topic/1688/constantes>

Capítulo 24: Constantes mágicas

Observaciones

Las constantes mágicas se distinguen por su forma `__CONSTANTNAME__`.

Actualmente hay ocho constantes mágicas que cambian dependiendo de dónde se usan. Por ejemplo, el valor de `__LINE__` depende de la línea que se usa en su script.

Estas constantes especiales no distinguen entre mayúsculas y minúsculas y son las siguientes:

Nombre	Descripción
<code>__LINE__</code>	El número de línea actual del archivo.
<code>__FILE__</code>	La ruta completa y el nombre del archivo con los enlaces simbólicos resueltos. Si se utiliza dentro de una inclusión, se devuelve el nombre del archivo incluido.
<code>__DIR__</code>	El directorio del archivo. Si se utiliza dentro de una inclusión, se devuelve el directorio del archivo incluido. Esto es equivalente a <code>dirname(__FILE__)</code> . Este nombre de directorio no tiene una barra inclinada a menos que sea el directorio raíz.
<code>__FUNCTION__</code>	El nombre de la función actual
<code>__CLASS__</code>	El nombre de la clase. El nombre de la clase incluye el espacio de nombres en el que se declaró (por ejemplo, <code>Foo\Bar</code>). Cuando se usa en un método de rasgo, <code>__CLASS__</code> es el nombre de la clase en que se usa el rasgo.
<code>__TRAIT__</code>	El nombre del rasgo. El nombre del rasgo incluye el espacio de nombres en el que se declaró (por ejemplo, <code>Foo\Bar</code>).
<code>__METHOD__</code>	El nombre del método de clase.
<code>__NAMESPACE__</code>	El nombre del espacio de nombres actual.

El caso de uso más común para estas constantes es la depuración y el registro.

Examples

Diferencia entre `__FUNCTION__` y `__METHOD__`

`__FUNCTION__` devuelve solo el nombre de la función, mientras que `__METHOD__` devuelve el nombre de la clase junto con el nombre de la función:

```

<?php

class trick
{
    public function doit()
    {
        echo __FUNCTION__;
    }

    public function doitagain()
    {
        echo __METHOD__;
    }
}

$obj = new trick();
$obj->doit(); // Outputs: doit
$obj->doitagain(); // Outputs: trick::doitagain

```

Diferencia entre __CLASS__, get_class () y get_called_class ()

__CLASS__ magic constant devuelve el mismo resultado que la función `get_class()` llamada sin parámetros y ambas devuelven el nombre de la clase donde se definió (es decir, donde escribió la función llamada / nombre constante).

Por el contrario, las `get_class($this)` y `get_called_class()` llaman, devolverán el nombre de la clase real de la que se creó una instancia:

```

<?php

class Definition_Class {

    public function say(){
        echo '__CLASS__ value: ' . __CLASS__ . "\n";
        echo 'get_called_class() value: ' . get_called_class() . "\n";
        echo 'get_class($this) value: ' . get_class($this) . "\n";
        echo 'get_class() value: ' . get_class() . "\n";
    }
}

class Actual_Class extends Definition_Class {}

$c = new Actual_Class();
$c->say();
// Output:
// __CLASS__ value: Definition_Class
// get_called_class() value: Actual_Class
// get_class($this) value: Actual_Class
// get_class() value: Definition_Class

```

Constantes de archivo y directorio

Archivo actual

Puede obtener el nombre del archivo PHP actual (con la ruta absoluta) utilizando la constante mágica `__FILE__`. Esto se utiliza más a menudo como una técnica de registro / depuración.

```
echo "We are in the file:" , __FILE__ , "\n";
```

Directorio actual

Para obtener la ruta absoluta al directorio donde se encuentra el archivo actual, use la constante mágica `__DIR__`.

```
echo "Our script is located in the:" , __DIR__ , "\n";
```

Para obtener la ruta absoluta al directorio donde se encuentra el archivo actual, use `dirname(__FILE__)`.

```
echo "Our script is located in the:" , dirname(__FILE__ ) , "\n";
```

Obtener el directorio actual a menudo es usado por marcos de PHP para establecer un directorio base:

```
// index.php of the framework  
  
define(BASEDIR, __DIR__); // using magic constant to define normal constant
```

```
// somefile.php looks for views:  
  
$view = 'page';  
$viewFile = BASEDIR . '/views/' . $view;
```

Separadores

El sistema de Windows entiende perfectamente las rutas / in, de modo que `DIRECTORY_SEPARATOR` se usa principalmente al analizar rutas.

Además de las constantes mágicas, PHP también agrega algunas constantes fijas para trabajar con rutas:

- Constante `DIRECTORY_SEPARATOR` para separar directorios en una ruta. Toma valor / en * nix, y \ en Windows. El ejemplo con vistas se puede reescribir con:

```
$view = 'page';  
$viewFile = BASEDIR . DIRECTORY_SEPARATOR . 'views' . DIRECTORY_SEPARATOR . $view;
```

- Rara vez se utiliza la constante `PATH_SEPARATOR` para separar las rutas en la `$PATH` entorno `$PATH`. Es ; en Windows, : de otro modo

Lea Constantes mágicas en línea: <https://riptutorial.com/es/php/topic/1428/constantes-magicas>

Capítulo 25: Contribuyendo al Manual de PHP

Introducción

El Manual de PHP proporciona una referencia funcional y una referencia de idioma junto con explicaciones de las principales funciones de PHP. El Manual de PHP, a diferencia de la documentación de muchos idiomas, alienta a los desarrolladores de PHP a que agreguen sus propios ejemplos y notas a cada página de la documentación. Este tema explica la contribución al manual de PHP, junto con consejos, trucos y pautas para las mejores prácticas.

Observaciones

Las contribuciones a este tema deben describir principalmente el proceso alrededor de la contribución al Manual de PHP, p. Ej., Explicar cómo agregar páginas, cómo enviarlas para su revisión, buscar áreas para contribuir con contenido, etc.

Examples

Mejorar la documentación oficial.

PHP ya tiene una gran documentación oficial en <http://php.net/manual/> . El Manual de PHP documenta prácticamente todas las características de idioma, las bibliotecas principales y las extensiones más disponibles. Hay muchos ejemplos para aprender. El Manual de PHP está disponible en múltiples idiomas y formatos.

Lo mejor de todo, **la documentación es gratuita para que cualquiera pueda editarla** .

El equipo de documentación de PHP proporciona un editor en línea para el Manual de PHP en <https://edit.php.net> . Admite múltiples servicios de inicio de sesión único, incluido el inicio de sesión con su cuenta de desbordamiento de pila. Puede encontrar una introducción al editor en <https://wiki.php.net/doc/editor> .

Los cambios en el Manual de PHP deben ser aprobados por personas del Equipo de Documentación de PHP que tenga *Doc Karma* . Doc Karma es algo así como reputación, pero más difícil de conseguir. Este proceso de revisión por pares se asegura de que solo la información correcta y objetiva entre en el Manual de PHP.

El Manual de PHP está escrito en DocBook, que es un lenguaje de marcado fácil de aprender para crear libros. Puede parecer un poco complicado a primera vista, pero hay plantillas para comenzar. Ciertamente no es necesario ser un experto en DocBook para contribuir.

Consejos para contribuir al manual.

La siguiente es una lista de consejos para aquellos que desean contribuir al manual de PHP:

- **Siga las pautas de estilo del manual** . Asegúrese de que las [pautas de estilo del manual](#) siempre se sigan por razones de coherencia.
- **Realizar correcciones ortográficas y gramaticales** . Asegúrese de que se esté utilizando la ortografía y la gramática adecuadas; de lo contrario, la información presentada puede ser más difícil de asimilar y el contenido se verá menos profesional.
- **Ser terso en las explicaciones** . Evite divagaciones para presentar de manera clara y concisa la información a los desarrolladores que buscan una referencia rápida.
- **Código separado de su salida** . Esto proporciona ejemplos de código más limpios y menos complejos para que los desarrolladores puedan digerirlos.
- **Compruebe el orden de la sección de la página** . Asegúrese de que todas las secciones de la página del manual que se está editando estén en el orden correcto. La uniformidad en el manual hace que sea más fácil leer y buscar información rápidamente.
- **Quitar contenido relacionado con PHP 4** . Las menciones específicas a PHP 4 ya no son relevantes, dada la antigüedad que tiene. Las menciones del mismo deben eliminarse del manual para evitar que se convulsa con información innecesaria.
- **Versión correcta de los archivos** . Al crear nuevos archivos en la documentación, asegúrese de que el ID de revisión del archivo esté configurado en nada, así: <!--
\$Revision\$ --> .
- **Fusionar comentarios útiles en el manual** . Algunos comentarios aportan información útil que el manual podría beneficiarse de tener. Estos deben ser combinados en el contenido de la página principal.
- **No rompas la compilación de documentación** . Asegúrese siempre de que el manual de PHP se compile correctamente antes de confirmar los cambios.

Lea Contribuyendo al Manual de PHP en línea:

<https://riptutorial.com/es/php/topic/2003/contribuyendo-al-manual-de-php>

Capítulo 26: Contribuyendo al PHP Core

Observaciones

PHP es un proyecto de código abierto, y como tal, cualquiera puede contribuir a él. En términos generales, hay dos formas de contribuir al núcleo de PHP:

- Corrección de errores
- Adiciones de características

Sin embargo, antes de contribuir, es importante comprender cómo se administran y liberan las versiones de PHP para que las correcciones de errores y las solicitudes de funciones puedan apuntar a la versión de PHP correcta. Los cambios desarrollados pueden enviarse como una solicitud de extracción al [repositorio de PHP Github](#). Puede encontrar información útil para los desarrolladores en la [sección "Involúcrese" del sitio PHP.net](#) y el [foro #externals](#).

Contribuyendo con la corrección de errores

Para aquellos que buscan comenzar a contribuir con el núcleo, generalmente es más fácil comenzar con la corrección de errores. Esto ayuda a familiarizarse con los elementos internos de PHP antes de intentar realizar modificaciones más complejas al núcleo que una característica requeriría.

Con respecto al proceso de administración de versiones, las correcciones de errores deben apuntar a las más afectadas, *mientras que aún se admite la versión de PHP*. Es esta versión a la que deben dirigirse las solicitudes de corrección de errores. Desde allí, un miembro interno puede combinar el arreglo en la rama correcta y luego combinarlo hacia arriba para obtener las versiones posteriores de PHP según sea necesario.

Para aquellos que buscan comenzar a resolver errores, puede encontrar una lista de informes de errores en [bugs.php.net](#).

Contribuyendo con adiciones de características

PHP sigue un proceso RFC cuando introduce nuevas funciones y realiza cambios importantes en el idioma. Los RFC son votados por miembros de php.net y deben alcanzar una mayoría simple (50% + 1) o una mayoría absoluta (2/3 + 1) del total de votos. Se requiere una mayoría absoluta si el cambio afecta al lenguaje en sí (como introducir una nueva sintaxis), de lo contrario solo se requiere una mayoría simple.

Antes de poder someter a votación las RFC, deben someterse a un período de discusión de al menos 2 semanas en la lista de correo oficial de PHP. Una vez que este período ha finalizado, y no hay problemas abiertos con el RFC, se puede mover a votación, que debe durar al menos 1 semana.

Si un usuario desea revivir un RFC rechazado anteriormente, puede hacerlo solo en una de las siguientes dos circunstancias:

- Han pasado 6 meses desde la votación anterior.
- El (los) autor (es) hacen cambios sustanciales en el RFC que probablemente afectarían el resultado de la votación en caso de que se vuelva a votar el RFC.

Las personas que tienen el privilegio de votar serán contribuyentes a PHP en sí (y por lo tanto tienen cuentas de `php.net`), o serán representantes de la comunidad de PHP. Estos representantes son elegidos por aquellos con cuentas de `php.net` y serán desarrolladores líderes de proyectos basados en PHP o participantes regulares para discusiones internas.

Al enviar nuevas ideas para una propuesta, casi siempre se requiere que el proponente escriba, como mínimo, un parche de prueba de concepto. Esto se debe a que sin una implementación, la sugerencia simplemente se convierte en otra solicitud de función que probablemente no se cumpla en un futuro cercano.

Puede encontrar una guía detallada de este proceso en la página oficial de [Cómo crear un RFC](#).

Lanzamientos

Las versiones principales de PHP no tienen un ciclo de lanzamiento establecido, por lo que pueden ser lanzadas a discreción del equipo interno (cuando lo consideren adecuado para un nuevo lanzamiento importante). Las versiones menores, por otro lado, se lanzan anualmente.

Antes de cada lanzamiento en PHP (mayor, menor o parche), una serie de candidatos de lanzamiento (RC) están disponibles. PHP no usa un RC como lo hacen otros proyectos (es decir, si un RC no tiene problemas para encontrarlo, entonces es el próximo lanzamiento final). En su lugar, los utiliza como una forma de betas finales, donde normalmente se decide un número determinado de RC antes de que se realice el lanzamiento final.

Versiones

PHP generalmente intenta seguir versiones semánticas siempre que sea posible. Como tal, la compatibilidad con versiones anteriores (BC) debe mantenerse en versiones secundarias y de parches del idioma. Las características y los cambios que conservan BC deben dirigirse a versiones menores (no a versiones de parches). Si una característica o cambio tiene el potencial de romper BC, entonces deberían apuntar a la siguiente versión importante de PHP (`X.yz`).

Cada versión menor de PHP (`x.Y.z`) tiene dos años de soporte general (denominado "soporte activo") para todos los tipos de correcciones de errores. Se agrega un año adicional a eso para soporte de seguridad, donde solo se aplican correcciones relacionadas con la seguridad. Después de los tres años, el soporte para esa versión de PHP se elimina por completo. Puede encontrar una lista de las [versiones de PHP actualmente soportadas en `php.net`](#).

Examples

Configuración de un entorno de desarrollo básico.

El código fuente de PHP está alojado en [GitHub](#). Para compilar desde la fuente, primero deberá retirar una copia de trabajo del código.

```
mkdir /usr/local/src/php-7.0/
cd /usr/local/src/php-7.0/
git clone -b PHP-7.0 https://github.com/php/php-src .
```

Si desea agregar una función, es mejor crear su propia sucursal.

```
git checkout -b my_private_branch
```

Finalmente, configura y construye PHP

```
./buildconf
./configure
make
make test
make install
```

Si la configuración falla debido a la falta de dependencias, deberá usar el sistema de administración de paquetes de su sistema operativo para instalarlos (por ejemplo, `yum`, `apt`, etc.) o descargarlos y compilarlos desde la fuente.

Lea Contribuyendo al PHP Core en línea: <https://riptutorial.com/es/php/topic/3929/contribuyendo-al-php-core>

Capítulo 27: Convenciones de codificación

Examples

Etiquetas PHP

Siempre debe usar etiquetas `<?php ?>` o etiquetas de eco corto `<?= ?>`. No se deben usar otras variaciones (en particular, etiquetas cortas `<? ?>`), ya que los administradores del sistema generalmente las deshabilitan.

Cuando no se espera que un archivo produzca una salida (¿todo el archivo es código PHP), se debe omitir la sintaxis `?>`. Para evitar una salida no intencional, lo que puede causar problemas cuando un cliente analiza el documento, en particular, algunos navegadores no reconocen el `<!DOCTYPE` y active el modo Quirks .

Ejemplo de un script PHP simple:

```
<?php  
  
print "Hello World";
```

Ejemplo de archivo de definición de clase:

```
<?php  
  
class Foo  
{  
    ...  
}
```

Ejemplo de PHP incrustado en HTML:

```
<ul id="nav">  
    <?php foreach ($navItems as $navItem): ?>  
        <li><a href="= htmlspecialchars($navItem-&gt;url) ?&gt;"&gt;<br/            <?= htmlspecialchars($navItem->label) ?>  
        </a></li>  
    <?php endforeach; ?>  
</ul>
```

Lea Convenciones de codificación en línea:

<https://riptutorial.com/es/php/topic/3977/convenciones-de-codificacion>

Capítulo 28: Corrientes

Sintaxis

- Cada flujo tiene un esquema y un objetivo:
- <esquema>: // <objetivo>

Parámetros

Nombre del parámetro	Descripción
Recurso de la corriente	El proveedor de datos que consiste en la sintaxis <scheme>://<target>

Observaciones

Las transmisiones son esencialmente una transferencia de datos entre un origen y un destino, parafraseando a Josh Lockhart en su libro Modern PHP.

El origen y el destino pueden ser

- un archivo
- un proceso de línea de comando
- una conexión de red
- un archivo ZIP o TAR
- memoria temporal
- entrada / salida estándar

o cualquier otro recurso disponible a través [de envolturas de flujo de PHP](#).

Ejemplos de envolturas de flujo disponibles (`schemes`):

- file: // - Accediendo al sistema de archivos local
- http: // - Accediendo a las URL de HTTP (s)
- ftp: // - Acceso a las URL de FTP (s)
- php: // - Accediendo a varias secuencias de E / S
- phar: // - PHP Archive
- ssh2: // - Secure Shell 2
- ogg: // - Transmisiones de audio

El esquema (origen) es el identificador de la envoltura del flujo. Por ejemplo, para el sistema de archivos, esto es `file://`. El destino es el origen de datos del flujo, por ejemplo, el nombre del archivo.

Examples

Registro de una envoltura de flujo

Una envoltura de flujo proporciona un controlador para uno o más esquemas específicos.

El siguiente ejemplo muestra un contenedor de flujo simple que envía solicitudes HTTP PATCH cuando se cierra el flujo.

```
// register the FooWrapper class as a wrapper for foo:// URLs.
stream_wrapper_register("foo", FooWrapper::class, STREAM_IS_URL) or die("Duplicate stream
wrapper registered");

class FooWrapper {
    // this will be modified by PHP to show the context passed in the current call.
    public $context;

    // this is used in this example internally to store the URL
    private $url;

    // when fopen() with a protocol for this wrapper is called, this method can be implemented
    // to store data like the host.
    public function stream_open(string $path, string $mode, int $options, string &$openedPath)
    : bool {
        $url = parse_url($path);
        if($url === false) return false;
        $this->url = $url["host"] . "/" . $url["path"];
        return true;
    }

    // handles calls to fwrite() on this stream
    public function stream_write(string $data) : int {
        $this->buffer .= $data;
        return strlen($data);
    }

    // handles calls to fclose() on this stream
    public function stream_close() {
        $curl = curl_init("http://" . $this->url);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $this->buffer);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PATCH");
        curl_exec($curl);
        curl_close($curl);
        $this->buffer = "";
    }

    // fallback exception handler if an unsupported operation is attempted.
    // this is not necessary.
    public function __call($name, $args) {
        throw new \RuntimeException("This wrapper does not support $name");
    }

    // this is called when unlink("foo://something-else") is called.
    public function unlink(string $path) {
        $url = parse_url($path);
        $curl = curl_init("http://" . $url["host"] . "/" . $url["path"]);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
        curl_exec($curl);
    }
}
```

```
    curl_close($curl);
}
}
```

Este ejemplo solo muestra algunos ejemplos de lo que contendría un contenedor de flujo genérico. Estos no son todos los métodos disponibles. Puede encontrar una lista completa de los métodos que se pueden implementar en <http://php.net/streamWrapper> .

Lea Corrientes en línea: <https://riptutorial.com/es/php/topic/5725/corrientes>

Capítulo 29: Crea archivos PDF en PHP

Examples

Empezando con PDFlib

Este código requiere que use la [biblioteca PDFlib](#) para que funcione correctamente.

```
<?php
$pdf = pdf_new(); //initialize new object

pdf_begin_document($pdf); //create new blank PDF
pdf_set_info($pdf, "Author", "John Doe"); //Set info about your PDF
pdf_set_info($pdf, "Title", "HelloWorld");
pdf_begin_page($pdf, (72 * 8.5), (72 * 11)); //specify page width and height
$font = pdf_findfont($pdf, "Times-Roman", "host", 0) //load a font
pdfSetFont($pdf, $font, 48); //set the font
pdf_set_text_pos($pdf, 50, 700); //assign text position
pdf_show($pdf, "Hello_World!"); //print text to assigned position
pdf_end_page($pdf); //end the page
pdf_end_document($pdf); //close the object

$document = pdf_get_buffer($pdf); //retrieve contents from buffer

$length = strlen($document); $filename = "HelloWorld.pdf"; //Finds PDF length and assigns file name

header("Content-Type:application/pdf");
header("Content-Length:" . $length);
header("Content-Disposition:inline; filename=" . $filename);

echo($document); //Send document to browser
unset($document); pdf_delete($pdf); //Clear Memory
?>
```

Lea [Crea archivos PDF en PHP en línea](#): <https://riptutorial.com/es/php/topic/4955/crea-archivos-pdf-en-php>

Capítulo 30: Criptografía

Observaciones

```
/* Base64 Encoded Encryption / $enc_data = base64_encode( openssl_encrypt($data, $method, $password, true, $iv) ); // Decode and Decrypt */ $dec_data = base64_decode( openssl_decrypt($enc_data, $method, $password, true, $iv) );
```

Esta forma de hacer el cifrado y la codificación no funcionaría tal como se presenta a medida que descifra el código antes de descodificar la base 64.

Necesitarías hacer esto en el orden opuesto.

```
/ This way instead / $enc_data=base64_encode(openssl_encrypt($data, $method, $pass, true, $iv)); $dec_data=openssl_decrypt(base64_decode($enc_data), $method, $pass, true, $iv);
```

Examples

Cifrado simétrico

Este ejemplo ilustra el cifrado simétrico AES 256 en modo CBC. Se necesita un vector de inicialización, así que generamos uno usando una función openssl. La variable `$strong` se usa para determinar si la IV generada fue criptográficamente fuerte.

Cifrado

```
$method = "aes-256-cbc"; // cipher method
$iv_length = openssl_cipher_iv_length($method); // obtain required IV length
$strong = false; // set to false for next line
$iv = openssl_random_pseudo_bytes($iv_length, $strong); // generate initialization vector

/* NOTE: The IV needs to be retrieved later, so store it in a database.
However, do not reuse the same IV to encrypt the data again. */

if(!$strong) { // throw exception if the IV is not cryptographically strong
    throw new Exception("IV not cryptographically strong!");
}

$data = "This is a message to be secured."; // Our secret message
$pass = "StackOverf10w"; // Our password

/* NOTE: Password should be submitted through POST over an HTTPS session.
Here, it's being stored in a variable for demonstration purposes. */

$enc_data = openssl_encrypt($data, $method, $pass, true, $iv); // Encrypt
```

Descifrado

```
/* Retrieve the IV from the database and the password from a POST request */
$dec_data = openssl_decrypt($enc_data, $method, $pass, true, $iv); // Decrypt
```

Base64 Codifica y Decodifica

Si los datos cifrados deben enviarse o almacenarse en texto imprimible, las funciones `base64_encode()` y `base64_decode()` deben usar respectivamente.

```
/* Base64 Encoded Encryption */
$enc_data = base64_encode(openssl_encrypt($data, $method, $password, true, $iv));

/* Decode and Decrypt */
$dec_data = openssl_decrypt(base64_decode($enc_data), $method, $password, true, $iv);
```

Cifrado y descifrado simétricos de archivos grandes con OpenSSL

PHP carece de una función incorporada para cifrar y descifrar archivos grandes. `openssl_encrypt` se puede usar para cifrar cadenas, pero cargar un archivo enorme en la memoria es una mala idea.

Así que tenemos que escribir una función de usuario haciendo eso. Este ejemplo utiliza el [algoritmo](#) simétrico [AES-128-CBC](#) para cifrar fragmentos más pequeños de un archivo grande y los escribe en otro archivo.

Cifrar archivos

```
/**
 * Define the number of blocks that should be read from the source file for each chunk.
 * For 'AES-128-CBC' each block consist of 16 bytes.
 * So if we read 10,000 blocks we load 160kb into memory. You may adjust this value
 * to read/write shorter or longer chunks.
 */
define('FILE_ENCRYPTION_BLOCKS', 10000);

/**
 * Encrypt the passed file and saves the result in a new file with ".enc" as suffix.
 *
 * @param string $source Path to file that should be encrypted
 * @param string $key     The key used for the encryption
 * @param string $dest    File name where the encrypted file should be written to.
 * @return string|false Returns the file name that has been created or FALSE if an error
occured
 */
function encryptFile($source, $key, $dest)
{
    $key = substr(sh1($key, true), 0, 16);
    $iv = openssl_random_pseudo_bytes(16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        // Put the initialization vector to the beginning of the file
        fwrite($fpOut, $iv);
```

```

if ($fpIn = fopen($source, 'rb')) {
    while (!feof($fpIn)) {
        $plaintext = fread($fpIn, 16 * FILE_ENCRYPTION_BLOCKS);
        $ciphertext = openssl_encrypt($plaintext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
        // Use the first 16 bytes of the ciphertext as the next initialization vector
        $iv = substr($ciphertext, 0, 16);
        fwrite($fpOut, $ciphertext);
    }
    fclose($fpIn);
} else {
    $error = true;
}
fclose($fpOut);
} else {
    $error = true;
}

return $error ? false : $dest;
}

```

Descifrar archivos

Para descifrar los archivos que se han cifrado con la función anterior, puede utilizar esta función.

```

/**
 * Decrypt the passed file and saves the result in a new file, removing the
 * last 4 characters from file name.
 *
 * @param string $source Path to file that should be decrypted
 * @param string $key      The key used for the decryption (must be the same as for encryption)
 * @param string $dest     File name where the decrypted file should be written to.
 * @return string|false Returns the file name that has been created or FALSE if an error
occurred
 */
function decryptFile($source, $key, $dest)
{
    $key = substr(sh1($key, true), 0, 16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        if ($fpIn = fopen($source, 'rb')) {
            // Get the initialization vector from the beginning of the file
            $iv = fread($fpIn, 16);
            while (!feof($fpIn)) {
                $ciphertext = fread($fpIn, 16 * (FILE_ENCRYPTION_BLOCKS + 1)); // we have to
read one block more for decrypting than for encrypting
                $plaintext = openssl_decrypt($ciphertext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $plaintext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    }
}

```

```
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Cómo utilizar

Si necesita un pequeño fragmento de código para ver cómo funciona esto o para probar las funciones anteriores, consulte el siguiente código.

```
$fileName = __DIR__.'/testfile.txt';
$key = 'my secret key';
file_put_contents($fileName, 'Hello World, here I am.');
encryptFile($fileName, $key, $fileName . '.enc');
decryptFile($fileName . '.enc', $key, $fileName . '.dec');
```

Esto creará tres archivos:

1. *testfile.txt* con el texto plano
2. *testfile.txt.enc* con el archivo encriptado
3. *testfile.txt.dec* con el archivo descifrado. Esto debería tener el mismo contenido que *testfile.txt*

Lea Criptografía en línea: <https://riptutorial.com/es/php/topic/5794/criptografia>

Capítulo 31: Datos de solicitud de lectura

Observaciones

Elegir entre GET y POST

Las solicitudes **GET** son las mejores para proporcionar los datos necesarios para representar la página y se pueden usar varias veces (consultas de búsqueda, filtros de datos ...). Son parte de la URL, lo que significa que se pueden marcar como favoritos y, a menudo, se reutilizan.

Las solicitudes **POST**, por otro lado, están destinadas a enviar datos al servidor solo una vez (formularios de contacto, formularios de inicio de sesión ...). A diferencia de GET, que solo acepta ASCII, las solicitudes POST también permiten datos binarios, incluidas las [cargas de archivos](#).

Puedes encontrar una explicación más detallada de sus diferencias [aquí](#).

Solicitar vulnerabilidades de datos

También mire: [¿cuáles son las vulnerabilidades en el uso directo de GET y POST?](#)

La recuperación de datos de las superglobales `$_GET` y `$_POST` sin ninguna validación se considera una mala práctica, y abre métodos para que los usuarios puedan acceder o comprometer datos a través de [inyecciones de código](#) y / o [SQL](#). Los datos no válidos deben ser revisados y rechazados para prevenir tales ataques.

Los datos de solicitud deben escaparse dependiendo de cómo se usa en el código, como se indica [aquí](#) y [aquí](#). En [esta respuesta](#) se pueden encontrar algunas funciones de escape diferentes para casos comunes de uso de datos.

Examples

Manejo de errores de carga de archivos

El `$_FILES["FILE_NAME"]['error']` (donde "FILE_NAME" es el valor del atributo de nombre de la entrada del archivo, presente en su formulario) puede contener uno de los siguientes valores:

1. `UPLOAD_ERR_OK` - No hay error, el archivo se cargó con éxito.
2. `UPLOAD_ERR_INI_SIZE` : el archivo cargado excede la directiva `upload_max_filesize` en `php.ini`.
3. `UPLOAD_ERR_PARTIAL` : el archivo cargado supera la directiva `MAX_FILE_SIZE` que se especificó en el formulario HTML.
4. `UPLOAD_ERR_NO_FILE` : no se ha cargado ningún archivo.
5. `UPLOAD_ERR_NO_TMP_DIR` - Falta una carpeta temporal. (Desde PHP 5.0.3).
6. `UPLOAD_ERR_CANT_WRITE` - Error al escribir el archivo en el disco. (Desde PHP 5.1.0).
7. `UPLOAD_ERR_EXTENSION` - Una extensión de PHP detuvo la carga del archivo. (Desde PHP 5.2.0).

Una forma básica de verificar los errores, es la siguiente:

```
<?php
$fileError = $_FILES["FILE_NAME"]["error"]; // where FILE_NAME is the name attribute of the
file input in your form
switch($fileError) {
    case UPLOAD_ERR_INI_SIZE:
        // Exceeds max size in php.ini
        break;
    case UPLOAD_ERR_PARTIAL:
        // Exceeds max size in html form
        break;
    case UPLOAD_ERR_NO_FILE:
        // No file was uploaded
        break;
    case UPLOAD_ERR_NO_TMP_DIR:
        // No /tmp dir to write to
        break;
    case UPLOAD_ERR_CANT_WRITE:
        // Error writing to disk
        break;
    default:
        // No error was faced! Phew!
        break;
}
```

Lectura de datos POST

Los datos de una solicitud POST se almacenan en el [superglobal](#) `$_POST` en forma de una matriz asociativa.

Tenga en cuenta que el acceso a un elemento de una matriz no existente genera un aviso, por lo que la existencia siempre debe verificarse con las `isset()` o `empty()`, o el operador de fusión nula.

Ejemplo:

```
$from = isset($_POST["name"]) ? $_POST["name"] : "NO NAME";
$message = isset($_POST["message"]) ? $_POST["message"] : "NO MESSAGE";

echo "Message from $from: $message";
```

7.0

```
$from = $_POST["name"] ?? "NO NAME";
$message = $_POST["message"] ?? "NO MESSAGE";

echo "Message from $from: $message";
```

Leyendo datos GET

Los datos de una solicitud GET se almacenan en el [superglobal](#) `$_GET` en forma de una matriz asociativa.

Tenga en cuenta que el acceso a un elemento de una matriz no existente genera un aviso, por lo

que la existencia siempre debe verificarse con las `isset()` o `empty()`, o el operador de fusión nula.

Ejemplo: (para URL `/topics.php?author=alice&topic=php`)

```
$author = isset($_GET["author"]) ? $_GET["author"] : "NO AUTHOR";
$topic = isset($_GET["topic"]) ? $_GET["topic"] : "NO TOPIC";

echo "Showing posts from $author about $topic";
```

7.0

```
$author = $_GET["author"] ?? "NO AUTHOR";
$topic = $_GET["topic"] ?? "NO TOPIC";

echo "Showing posts from $author about $topic";
```

Lectura de datos POST sin procesar

Generalmente, los datos enviados en una solicitud POST son pares de clave / valor estructurados con un tipo MIME de `application/x-www-form-urlencoded`. Sin embargo, muchas aplicaciones, como los servicios web, requieren que se envíen datos sin procesar, a menudo en formato XML o JSON. Estos datos se pueden leer usando uno de dos métodos.

`php://input` es una secuencia que proporciona acceso al cuerpo de solicitud sin formato.

```
$rawdata = file_get_contents("php://input");
// Let's say we got JSON
$decoded = json_decode($rawdata);
```

5.6

`$HTTP_RAW_POST_DATA` es una variable global que contiene los datos POST sin procesar. Solo está disponible si la directiva `always_populate_raw_post_data` en `php.ini` está habilitada.

```
$rawdata = $HTTP_RAW_POST_DATA;
// Or maybe we get XML
$decoded = simplexml_load_string($rawdata);
```

Esta variable ha quedado en desuso desde la versión 5.6 de PHP y se eliminó en PHP 7.0.

Tenga en cuenta que ninguno de estos métodos está disponible cuando el tipo de contenido se establece en `multipart/form-data`, que se utiliza para cargar archivos.

Subiendo archivos con HTTP PUT

PHP proporciona soporte para el método HTTP PUT utilizado por algunos clientes para almacenar archivos en un servidor. Las solicitudes PUT son mucho más simples que una carga de archivos usando solicitudes POST y se ven algo así:

```
PUT /path/filename.html HTTP/1.1
```

En tu código PHP entonces harías algo como esto:

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://input", "r");

/* Open a file for writing */
$fp = fopen("putfile.ext", "w");

/* Read the data 1 KB at a time
   and write to the file */
while ($data = fread($putdata, 1024))
    fwrite($fp, $data);

/* Close the streams */
fclose($fp);
fclose($putdata);
?>
```

También [aquí](#) puede leer interesantes preguntas / respuestas SO sobre recibir archivos a través de HTTP PUT.

Pasando matrices por POST

Generalmente, un elemento de formulario HTML enviado a PHP da como resultado un solo valor. Por ejemplo:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo" value="bar"/>
    <button type="submit">Submit</button>
</form>
```

Esto resulta en el siguiente resultado:

```
Array
(
    [foo] => bar
)
```

Sin embargo, puede haber casos en los que desee pasar una matriz de valores. Esto se puede hacer agregando un sufijo similar a PHP al nombre de los elementos HTML:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[]" value="bar"/>
    <input type="hidden" name="foo[]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Esto resulta en el siguiente resultado:

```
Array
(
    [foo] => Array
        (
            [0] => bar
            [1] => baz
        )
)
```

También puede especificar los índices de matriz, como números o cadenas:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[42]" value="bar"/>
    <input type="hidden" name="foo[foo]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Lo que devuelve esta salida:

```
Array
(
    [foo] => Array
        (
            [42] => bar
            [foo] => baz
        )
)
```

Esta técnica se puede usar para evitar lapsos de post-procesamiento sobre la matriz `$_POST`, haciendo que su código sea más simple y conciso.

Lea Datos de solicitud de lectura en línea: <https://riptutorial.com/es/php/topic/2668/datos-de-solicitud-de-lectura>

Capítulo 32: Depuración

Examples

Variables de dumping

La función `var_dump` permite volcar el contenido de una variable (tipo y valor) para la depuración.

Ejemplo:

```
$array = [3.7, "string", 10, ["hello" => "world"], false, new DateTime()];  
var_dump($array);
```

Salida:

```
array(6) {  
    [0]=>  
    float(3.7)  
    [1]=>  
    string(6) "string"  
    [2]=>  
    int(10)  
    [3]=>  
    array(1) {  
        ["hello"]=>  
        string(5) "world"  
    }  
    [4]=>  
    bool(false)  
    [5]=>  
    object(DateTime)#1 (3) {  
        ["date"]=>  
        string(26) "2016-07-24 13:51:07.000000"  
        ["timezone_type"]=>  
        int(3)  
        ["timezone"]=>  
        string(13) "Europe/Berlin"  
    }  
}
```

Mostrando errores

Si desea que PHP muestre errores de tiempo de ejecución en la página, debe habilitar `display_errors`, ya sea en `php.ini` o usando la función `ini_set`.

Puede elegir qué errores mostrar, con la función `error_reporting` (o `ini`), que acepta las constantes `E_*`, combinadas usando `operadores bitwise`.

PHP puede mostrar errores en formato de texto o HTML, dependiendo de la configuración `html_errors`.

Ejemplo:

```
ini_set("display_errors", true);
ini_set("html_errors", false); // Display errors in plain text
error_reporting(E_ALL & ~E_USER_NOTICE); // Display everything except E_USER_NOTICE

trigger_error("Pointless error"); // E_USER_NOTICE
echo $nonexistentVariable; // E_NOTICE
nonexistentFunction(); // E_ERROR
```

Salida de texto sin formato : (el formato HTML difiere entre las implementaciones)

```
Notice: Undefined variable: nonexistentVariable in /path/to/file.php on line 7

Fatal error: Uncaught Error: Call to undefined function nonexistentFunction() in
/path/to/file.php:8
Stack trace:
#0 {main}
    thrown in /path/to/file.php on line 8
```

NOTA: Si tiene el informe de errores deshabilitado en php.ini y lo habilita durante el tiempo de ejecución, algunos errores (como los errores de análisis) no se mostrarán, ya que ocurrieron antes de que se aplicara la configuración de tiempo de ejecución.

La forma común de manejar `error_reporting` es habilitarlo completamente con la constante `E_ALL` durante el desarrollo, y deshabilitar mostrarlo públicamente con `display_errors` en la etapa de producción para ocultar las partes internas de sus scripts.

`phpinfo()`

Advertencia

Es imperativo que `phpinfo` solo se use en un entorno de desarrollo. Nunca libere el código que contiene `phpinfo` en un entorno de producción

Introducción

Dicho esto, puede ser una herramienta útil para comprender el entorno PHP (SO, configuración, versiones, rutas, módulos) en el que está trabajando, especialmente cuando persigue un error. Es una simple función incorporada:

```
phpinfo();
```

Tiene un parámetro `$what` que permite personalizar la salida. El valor predeterminado es `INFO_ALL`, lo que hace que muestre toda la información y se usa comúnmente durante el desarrollo para ver el estado actual de PHP.

Puede pasar las **constantes** del parámetro `INFO_*`, combinadas con operadores bitwise para ver

una lista personalizada.

Puede ejecutarlo en el navegador para una apariencia detallada bien formateada. También funciona en PHP CLI, donde puede canalizarlo a `less` para una vista más fácil.

Ejemplo

```
phpinfo(INFO_CONFIGURATION | INFO_ENVIRONMENT | INFO_VARIABLES);
```

Esto mostrará una lista de directivas de PHP (`ini_get`), entorno (`$_ENV`) y variables [predefinidas](#).

Xdebug

Xdebug es una extensión de PHP que proporciona capacidades de depuración y creación de perfiles.

Utiliza el protocolo de depuración DBGp.

Hay algunas características agradables en esta herramienta:

- apilar trazas de errores
- Máxima protección de nivel de anidación y seguimiento del tiempo.
- Reemplazo útil de la función `var_dump()` estándar para mostrar variables
- permite registrar todas las llamadas de función, incluidos los parámetros y devolver valores a un archivo en diferentes formatos
- análisis de cobertura de código
- información de perfil
- depuración remota (proporciona una interfaz para los clientes del depurador que interactúan con scripts PHP en ejecución)

Como puede ver, esta extensión es perfectamente adecuada para el entorno de desarrollo.

Especialmente la función de **depuración remota** puede ayudarlo a depurar su código php sin numerosos `var_dump` y usar el proceso de depuración normal como en los `C++` o `Java`.

Generalmente la instalación de esta extensión es muy simple:

```
pecl install xdebug # install from pecl/pear
```

Y activalo en tu `php.ini`:

```
zend_extension="/usr/local/php/modules/xdebug.so"
```

En casos más complicados ver estas [instrucciones](#).

Cuando uses esta herramienta debes recordar que:

XDebug no es adecuado para entornos de producción

phpversion ()

Introducción

Cuando se trabaja con varias bibliotecas y sus requisitos asociados, a menudo es necesario conocer la versión del analizador PHP actual o uno de sus paquetes.

Esta función acepta un único parámetro opcional en forma de nombre de extensión:

`phpversion('extension')` . Si la extensión en cuestión está instalada, la función devolverá una cadena que contiene el valor de la versión. Sin embargo, si la extensión no instalada `FALSE` será devuelta. Si no se proporciona el nombre de la extensión, la función devolverá la versión del propio analizador PHP.

Ejemplo

```
print "Current PHP version: " . phpversion();
// Current PHP version: 7.0.8

print "Current cURL version: " . phpversion( 'curl' );
// Current cURL version: 7.0.8
// or
// false, no printed output if package is missing
```

Informe de errores (utilícelos ambos)

```
// this sets the configuration option for your environment
ini_set('display_errors', '1');

// -1 will allow all errors to be reported
error_reporting(-1);
```

Lea Depuración en línea: <https://riptutorial.com/es/php/topic/3339/depuracion>

Capítulo 33: Despliegue de Docker

Introducción

Docker es una solución de contenedor muy popular que se usa ampliamente para implementar código en entornos de producción. Facilita la *administración* y *escalado* de aplicaciones web y microservicios.

Observaciones

Este documento asume que la ventana acopiable está instalada y el daemon en ejecución. Puede consultar la [instalación de Docker](#) para verificar cómo instalar la misma.

Examples

Obtener imagen docker para php

Para implementar la aplicación en la ventana acopiable, primero necesitamos obtener la imagen del registro.

```
docker pull php
```

Esto te dará la última versión de la imagen del *repositorio oficial de php*. En términos generales, PHP generalmente se usa para implementar aplicaciones web, por lo que necesitamos un servidor http para ir con la imagen. La imagen de php:7.0-apache viene preinstalada con apache para que la implementación sea sencilla.

Escritura dockerfile

Dockerfile se utiliza para configurar la imagen personalizada que construiremos con los códigos de la aplicación web. Cree un nuevo archivo Dockerfile en la carpeta raíz del proyecto y luego coloque los siguientes contenidos en el mismo

```
FROM php:7.0-apache
COPY /etc/php/php.ini /usr/local/etc/php/
COPY . /var/www/html/
EXPOSE 80
```

La primera línea es bastante sencilla y se usa para describir qué imagen se debe usar para construir una nueva imagen. Lo mismo podría cambiarse a cualquier otra versión específica de PHP desde el registro.

La segunda línea es simplemente subir el archivo php.ini a nuestra imagen. Siempre puede cambiar ese archivo a otra ubicación de archivo personalizado.

La tercera línea copiaría los códigos en el directorio actual a `/var/www/html` que es nuestro webroot. Recuerda `/var/www/html` dentro de la imagen.

La última línea simplemente abriría el puerto 80 dentro del contenedor de la ventana acopiable.

Ignorando archivos

En algunos casos, es posible que haya algunos archivos que no desea en el servidor, como la configuración del entorno, etc. Supongamos que tenemos nuestro entorno en `.env`. Ahora, para ignorar este archivo, simplemente podemos agregarlo a `.dockerignore` en la carpeta raíz de nuestro código base.

Imagen del edificio

Crear una imagen no es algo específico de `php`, pero para construir la imagen que describimos anteriormente, simplemente podemos usar

```
docker build -t <Image name> .
```

Una vez que la imagen está construida, puedes verificar la misma usando

```
docker images
```

Lo que enumera todas las imágenes instaladas en su sistema.

Iniciar contenedor de aplicaciones

Una vez que tengamos una imagen lista, podemos comenzar y servir la misma. Para crear un `container` partir de la imagen, use

```
docker run -p 80:80 -d <Image name>
```

En el comando anterior, `-p 80:80` reenvía el puerto 80 de su servidor al puerto 80 del contenedor. La bandera `-d` indica que el contenedor debe ejecutarse como trabajo de fondo. El final especifica qué imagen debe usarse para construir el contenedor.

Contenedor de control

Para verificar los contenedores en funcionamiento, simplemente use

```
docker ps
```

Esto mostrará una lista de todos los contenedores que se ejecutan en el demonio docker.

Registros de aplicación

Los registros son muy importantes para depurar la aplicación. Para comprobar su uso.

```
docker logs <Container id>
```

Lea Despliegue de Docker en línea: <https://riptutorial.com/es/php/topic/9327/despliegue-de-docker>

Capítulo 34: DOP

Introducción

La extensión **PDO** (PHP Data Objects) permite a los desarrolladores conectarse a numerosos tipos diferentes de bases de datos y ejecutar consultas contra ellos de manera uniforme y orientada a objetos.

Sintaxis

- `PDO::LastInsertId()`
- `PDO::LastInsertId($columnName)` // algunos controladores necesitan el nombre de columna

Observaciones

Advertencia No deje de verificar las excepciones mientras usa `lastInsertId()`. Puede lanzar el siguiente error:

SQLSTATE IM001: el controlador no admite esta función

Aquí es cómo debe verificar adecuadamente las excepciones utilizando este método:

```
// Retrieving the last inserted id
$id = null;

try {
    $id = $pdo->lastInsertId(); // return value is an integer
}
catch( PDOException $e ) {
    echo $e->getMessage();
}
```

Examples

Conexión y recuperación de PDO básica

Desde PHP 5.0, **PDO** ha estado disponible como una capa de acceso a la base de datos. Es independiente de la base de datos, por lo que el siguiente código de conexión debería funcionar para cualquiera [de sus bases de datos compatibles](#) simplemente cambiando el DSN.

```
// First, create the database handle

//Using MySQL (connection via local socket):
$dsn = "mysql:host=localhost;dbname=testdb;charset=utf8";

//Using MySQL (connection via network, optionally you can specify the port too):
// $dsn = "mysql:host=127.0.0.1;port=3306;dbname=testdb;charset=utf8";
```

```

//Or Postgres
//$dsn = "pgsql:host=localhost;port=5432;dbname=testdb;";

//Or even SQLite
//$dsn = "sqlite:/path/to/database"

$username = "user";
$password = "pass";
$db = new PDO($dsn, $username, $password);

// setup PDO to throw an exception if an invalid query is provided
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Next, let's prepare a statement for execution, with a single placeholder
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);

// Create some parameters to fill the placeholders, and execute the statement
$parameters = [ "221B" ];
$statement->execute($parameters);

// Now, loop through each record as an associative array
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}

```

La función de `prepare` crea un objeto `PDOStatement` partir de la cadena de consulta. La ejecución de la consulta y la recuperación de los resultados se realizan en este objeto devuelto. En caso de una falla, la función devuelve `false` o lanza una `exception` (dependiendo de cómo se configuró la conexión PDO).

Prevención de la inyección SQL con consultas parametrizadas

La inyección SQL es un tipo de ataque que permite a un usuario malintencionado modificar la consulta SQL, agregándole comandos no deseados. Por ejemplo, el siguiente código es **vulnerable** :

```

// Do not use this vulnerable code!
$sql = 'SELECT name, email, user_level FROM users WHERE userID = ' . $_GET['user'];
$conn->query($sql);

```

Esto permite a cualquier usuario de este script modificar nuestra base de datos básicamente a voluntad. Por ejemplo, considere la siguiente cadena de consulta:

```
page.php?user=0;%20TRUNCATE%20TABLE%20users;
```

Esto hace que nuestra consulta de ejemplo se vea así.

```
SELECT name, email, user_level FROM users WHERE userID = 0; TRUNCATE TABLE users;
```

Si bien este es un ejemplo extremo (la mayoría de los ataques de inyección de SQL no pretenden eliminar datos, ni la mayoría de las funciones de ejecución de consultas de PHP son compatibles

con las consultas múltiples), este es un ejemplo de cómo un ataque de inyección de SQL puede ser posible por el montaje descuidado de la consulta. Desafortunadamente, los ataques de este tipo son muy comunes y son altamente efectivos debido a los codificadores que no toman las precauciones adecuadas para proteger sus datos.

Para evitar que se produzca la inyección de SQL, las **declaraciones preparadas** son la solución recomendada. En lugar de concatenar datos de usuario directamente a la consulta, se utiliza un *marcador de posición* en su lugar. Los datos se envían por separado, lo que significa que no hay posibilidad de que el motor de SQL confunda los datos del usuario para un conjunto de instrucciones.

Si bien el tema aquí es PDO, tenga en cuenta que la extensión MySQLi de PHP también [admite declaraciones preparadas](#)

PDO admite dos tipos de marcadores de posición (los marcadores de posición no se pueden usar para nombres de columnas o tablas, solo valores):

1. Nombrados marcadores de posición. Un colon (:), seguido por un nombre distinto (por ejemplo. :user)

```
// using named placeholders
$sql = 'SELECT name, email, user_level FROM users WHERE userID = :user';
$prep = $conn->prepare($sql);
$prep->execute(['user' => $_GET['user']]);
$result = $prep->fetchAll();
```

2. Marcadores de posición posicionales de SQL tradicionales, representados como ? :

```
// using question-mark placeholders
$sql = 'SELECT name, user_level FROM users WHERE userID = ? AND user_level = ?';
$prep = $conn->prepare($sql);
$prep->execute($_GET['user'], $_GET['user_level']);
$result = $prep->fetchAll();
```

Si alguna vez necesita cambiar dinámicamente los nombres de tablas o columnas, sepia que esto es responsabilidad de su propio riesgo y una mala práctica. Sin embargo, se puede hacer por concatenación de cuerdas. Una forma de mejorar la seguridad de dichas consultas es establecer una tabla de valores permitidos y comparar el valor que desea concatenar con esta tabla.

Tenga en cuenta que es importante establecer el conjunto de caracteres de la conexión solo a través de DSN, de lo contrario, su aplicación podría estar expuesta a una [vulnerabilidad poco clara](#) si se utiliza alguna codificación impar. Para versiones PDO anteriores a 5.3.6, la configuración del conjunto de caracteres a través de DSN no está disponible y, por lo tanto, la única opción es establecer el atributo `PDO::ATTR_EMULATE_PREPARES` en `false` en la conexión inmediatamente después de su creación.

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

Esto hace que PDO utilice las declaraciones preparadas nativas del DBMS subyacente en lugar

de simplemente emularlo.

Sin embargo, tenga en cuenta que la DOP [retrocederá silenciosamente](#) para emular las declaraciones que MySQL no puede preparar de forma nativa: las que pueden [aparecer en el manual](#) ([fuente](#)).

DOP: conexión a servidor MySQL / MariaDB

Hay dos formas de conectarse a un servidor MySQL / MariaDB, dependiendo de su infraestructura.

Conexión estándar (TCP / IP)

```
$dsn = 'mysql:dbname=demo;host=server;port=3306;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

Dado que PDO fue diseñado para ser compatible con versiones anteriores del servidor MySQL (que no tenía soporte para sentencias preparadas), debe deshabilitar explícitamente la emulación. De lo contrario, perderá los beneficios adicionales de **prevención de inyecciones**, que generalmente se otorgan utilizando declaraciones preparadas.

Otro compromiso de diseño, que debe tener en cuenta, es el comportamiento de manejo de errores predeterminado. Si no se configura de otra manera, PDO no mostrará ninguna indicación de errores de SQL.

Se recomienda encarecidamente configurarlo en "modo de excepción", porque eso le otorga una funcionalidad adicional al escribir abstracciones de persistencia (por ejemplo: tener una excepción, al violar la restricción `UNIQUE`).

Conexión de zócalo

```
$dsn = 'mysql:unix_socket=/tmp/mysql.sock;dbname=demo;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

En sistemas similares a Unix, si el nombre de host es '`localhost`', entonces la conexión al servidor se realiza a través de un socket de dominio.

Transacciones de base de datos con DOP

Las transacciones de la base de datos aseguran que un conjunto de cambios en los datos solo se harán permanentes si cada declaración es exitosa. Se puede detectar cualquier error de consulta o código durante una transacción y, a continuación, tiene la opción de revertir los cambios intentados.

La DOP proporciona métodos simples para iniciar, confirmar y revertir transacciones.

```
$pdo = new PDO(
    $dsn,
    $username,
    $password,
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
);

try {
    $statement = $pdo->prepare("UPDATE user SET name = :name");
    $pdo->beginTransaction();
    $statement->execute(["name"=>'Bob']);
    $statement->execute(["name"=>'Joe']);

    $pdo->commit();
}
catch (\Exception $e) {
    if ($pdo->inTransaction()) {
        $pdo->rollback();
        // If we got here our two data updates are not in the database
    }
    throw $e;
}
```

Durante una transacción, los cambios de datos realizados solo son visibles para la conexión activa. `SELECT` instrucciones `SELECT` devolverán los cambios modificados incluso si aún no están comprometidos con la base de datos.

Nota : consulte la documentación del proveedor de la base de datos para obtener detalles sobre el soporte de transacciones. Algunos sistemas no admiten transacciones en absoluto. Algunos admiten transacciones anidadas, mientras que otros no.

Ejemplo práctico del uso de transacciones con DOP

En la siguiente sección se muestra un ejemplo práctico del mundo real donde el uso de transacciones garantiza la consistencia de la base de datos.

Imagine el siguiente escenario, digamos que está construyendo un carrito de compras para un sitio web de comercio electrónico y decidió mantener los pedidos en dos tablas de base de datos. Una `orders` nombrada con los campos `order_id`, `name`, `address`, `telephone` y `created_at`. Y una segunda llamada `orders_products` con los campos `order_id`, `product_id` y `quantity`. La primera tabla contiene los **metadatos** del pedido, mientras que la segunda contiene los **productos** reales que se han pedido.

Inserción de un nuevo pedido en la base de datos.

Para insertar un nuevo pedido en la base de datos debe hacer dos cosas. Primero necesita `INSERT` un nuevo registro dentro de la tabla de `orders` que contendrá los **metadatos** del pedido (`name`, `address`, etc.). Y luego necesita `INSERT` un registro en la tabla `orders_products`, para cada uno de los productos que se incluyen en el pedido.

Puedes hacer esto haciendo algo similar a lo siguiente:

```
// Insert the metadata of the order into the database
$preparedStatement = $db->prepare(
    'INSERT INTO `orders` (`name`, `address`, `telephone`, `created_at`)
        VALUES (:name, :address, :telephone, :created_at)'
);

$preparedStatement->execute([
    'name' => $name,
    'address' => $address,
    'telephone' => $telephone,
    'created_at' => time(),
]);

// Get the generated `order_id`
$orderId = $db->lastInsertId();

// Construct the query for inserting the products of the order
$insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

$count = 0;
foreach ($products as $productId => $quantity) {
    $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity'
    . $count . ')';
}

$insertProductsParams['order_id' . $count] = $orderId;
$insertProductsParams['product_id' . $count] = $productId;
$insertProductsParams['quantity' . $count] = $quantity;

++$count;
}

// Insert the products included in the order into the database
$preparedStatement = $db->prepare($insertProductsQuery);
$preparedStatement->execute($insertProductsParams);
```

Esto funcionará muy bien para insertar un nuevo pedido en la base de datos, hasta que ocurra algo inesperado y, por alguna razón, la segunda consulta `INSERT` falle. Si eso sucede, terminará con un nuevo pedido dentro de la tabla de `orders`, que no tendrá productos asociados. Afortunadamente, la solución es muy simple, todo lo que tiene que hacer es hacer las consultas en forma de una sola transacción de base de datos.

Inserción de un nuevo pedido en la base de datos con una transacción

Para iniciar una transacción utilizando `PDO` todo lo que tiene que hacer es llamar al método `beginTransaction` antes de ejecutar cualquier consulta en su base de datos. Luego, realice los cambios que desee en sus datos ejecutando las consultas `INSERT` y / o `UPDATE`. Y, finalmente, se

llama al método `commit` del objeto `PDO` para que los cambios sean permanentes. Hasta que llame al método de `commit`, todos los cambios que haya realizado en sus datos hasta este momento aún no son permanentes, y pueden revertirse fácilmente simplemente llamando al método de `rollback` del objeto `PDO`.

En el siguiente ejemplo, se muestra el uso de transacciones para insertar un nuevo pedido en la base de datos, al tiempo que se garantiza la coherencia de los datos. Si una de las dos consultas falla, todos los cambios serán revertidos.

```
// In this example we are using MySQL but this applies to any database that has support for
transactions
$db = new PDO('mysql:host=' . $host . ';dbname=' . $dbname . ';charset=utf8', $username,
$password);

// Make sure that PDO will throw an exception in case of error to make error handling easier
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // From this point and until the transaction is being committed every change to the
database can be reverted
    $db->beginTransaction();

    // Insert the metadata of the order into the database
    $preparedStatement = $db->prepare(
        'INSERT INTO `orders` (`order_id`, `name`, `address`, `created_at`)
        VALUES (:name, :address, :telephone, :created_at)'
    );

    $preparedStatement->execute([
        'name' => $name,
        'address' => $address,
        'telephone' => $telephone,
        'created_at' => time(),
    ]);

    // Get the generated `order_id`
    $orderId = $db->lastInsertId();

    // Construct the query for inserting the products of the order
    $insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`,
`quantity`) VALUES';

    $count = 0;
    foreach ($products as $productId => $quantity) {
        $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ',
:quantity' . $count . ')';

        $insertProductsParams['order_id' . $count] = $orderId;
        $insertProductsParams['product_id' . $count] = $productId;
        $insertProductsParams['quantity' . $count] = $quantity;

        ++$count;
    }

    // Insert the products included in the order into the database
    $preparedStatement = $db->prepare($insertProductsQuery);
    $preparedStatement->execute($insertProductsParams);

    // Make the changes to the database permanent
```

```

    $db->commit();
}
catch ( PDOException $e ) {
    // Failed to insert the order into the database so we rollback any changes
    $db->rollback();
    throw $e;
}

```

DOP: obtener el número de filas afectadas por una consulta

Comenzamos con `$db`, una instancia de la clase PDO. Después de ejecutar una consulta, a menudo queremos determinar el número de filas que se han visto afectadas por ella. El método `rowCount()` de `PDOStatement` funcionará bien:

```

$query = $db->query("DELETE FROM table WHERE name = 'John'");
$count = $query->rowCount();

echo "Deleted $count rows named John";

```

NOTA: este método solo se debe usar para determinar el número de filas afectadas por las instrucciones INSERT, DELETE y UPDATE. Aunque este método también puede funcionar para sentencias SELECT, no es consistente en todas las bases de datos.

DOP :: lastInsertId ()

A menudo puede encontrar la necesidad de obtener el valor de ID incrementado automáticamente para una fila que acaba de insertar en su tabla de base de datos. Puedes lograr esto con el método `lastInsertId()`.

```

// 1. Basic connection opening (for MySQL)
$host = 'localhost';
$database = 'foo';
$user = 'root';
$password = '';
$dsn = "mysql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')";
$query_success = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $pdo->lastInsertId(); // return value is an integer

```

En postgresql y oracle, está la palabra clave RETURNING, que devuelve las columnas especificadas de las filas actualmente insertadas / modificadas. Aquí el ejemplo para insertar una entrada:

```

// 1. Basic connection opening (for PGSQL)
$host = 'localhost';
$database = 'foo';
$user = 'root';
$password = '';

```

```
$dsn = "pgsql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')
RETURNING id";
$statement = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $statement->fetchColumn(); // return the value of the id column of the new row in
foo_user
```

Lea DOP en línea: <https://riptutorial.com/es/php/topic/5828/dop>

Capítulo 35: Ejecutando sobre una matriz

Examples

Aplicando una función a cada elemento de una matriz.

Para aplicar una función a cada elemento de una matriz, use `array_map()`. Esto devolverá una nueva matriz.

```
$array = array(1,2,3,4,5);
//each array item is iterated over and gets stored in the function parameter.
$newArray = array_map(function($item) {
    return $item + 1;
}, $array);
```

\$newArray ahora es `array(2,3,4,5,6)`.

En lugar de usar una [función anónima](#), podría usar una función nombrada. Lo anterior podría escribirse como:

```
function addOne($item) {
    return $item + 1;
}

$array = array(1, 2, 3, 4, 5);
$newArray = array_map('addOne', $array);
```

Si la función nombrada es un método de clase, la llamada de la función debe incluir una referencia a un objeto de clase al que pertenece el método:

```
class Example {
    public function addOne($item) {
        return $item + 1;
    }

    public function doCalculation() {
        $array = array(1, 2, 3, 4, 5);
        $newArray = array_map(array($this, 'addOne'), $array);
    }
}
```

Otra forma de aplicar una función a cada elemento de una matriz es `array_walk()` y `array_walk_recursive()`. La devolución de llamada pasada a estas funciones toma tanto la clave / índice como el valor de cada elemento de la matriz. Estas funciones no devolverán una nueva matriz, en lugar de un booleano para el éxito. Por ejemplo, para imprimir cada elemento en una matriz simple:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function($value, $key) {
    echo $value . ' ';
```

```
});  
// prints "1 2 3 4 5"
```

El parámetro de valor de la devolución de llamada se puede pasar por referencia, lo que le permite cambiar el valor directamente en la matriz original:

```
$array = array(1, 2, 3, 4, 5);  
array_walk($array, function(&$value, $key) {  
    $value++;  
});
```

\$array ahora es array(2, 3, 4, 5, 6);

Para matrices anidadas, `array_walk_recursive()` irá más profundo en cada sub-matriz:

```
$array = array(1, array(2, 3, array(4, 5), 6);  
array_walk_recursive($array, function($value, $key) {  
    echo $value . ' ';  
});  
// prints "1 2 3 4 5 6"
```

Nota : `array_walk` y `array_walk_recursive` permiten cambiar el valor de los elementos de la matriz, pero no las claves. Pasar las claves por referencia a la devolución de llamada es válido pero no tiene ningún efecto.

Dividir matriz en trozos

`array_chunk ()` divide una matriz en trozos

Digamos que estamos siguiendo una matriz unidimensional,

```
$input_array = array('a', 'b', 'c', 'd', 'e');
```

Ahora usando `array_chunk ()` en la matriz de PHP anterior,

```
$output_array = array_chunk($input_array, 2);
```

El código anterior creará fragmentos de 2 elementos de matriz y creará una matriz multidimensional como sigue.

```
Array  
(  
 [0] => Array  
     (  
         [0] => a  
         [1] => b  
     )  
  
 [1] => Array  
     (  
         [0] => c
```

```

        [1] => d
    )

[2] => Array
(
    [0] => e
)
)

```

Si todos los elementos de la matriz no están divididos equitativamente por el tamaño del fragmento, el último elemento de la matriz de salida serán los elementos restantes.

Si pasamos el segundo argumento como menos de 1, se **lanzará E_WARNING** y la matriz de salida será **NULL**.

Parámetro	Detalles
\$array (array)	Matriz de entrada, la matriz para trabajar en
\$size (int)	Tamaño de cada trozo (valor entero)
\$preserve_keys (boolean) (opcional)	Si desea que la matriz de salida conserve las claves, establézcala en VERDADERO de lo contrario FALSO .

Implementando una matriz en una cadena

`implode()` combina todos los valores de la matriz pero pierde toda la información clave:

```

$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", $arr); // AA BB CC

```

Las claves de `array_keys()` se pueden hacer usando `array_keys()` call:

```

$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_keys($arr)); // a b c

```

Implodificar claves con valores es más complejo, pero se puede hacer usando un estilo funcional:

```

$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_map(function($key, $val) {
    return "$key:$val"; // function that glues key to the value
}, array_keys($arr), $arr));

// Output: a:AA b:BB c:CC

```

array_reduce

`array_reduce` reduce la matriz en un solo valor. Básicamente, el `array_reduce` pasará por cada elemento con el resultado de la última iteración y producirá un nuevo valor para la siguiente iteración.

Uso: `array_reduce ($array, function($carry, $item){...}, $default_value_of_first_carry)`

- `$ carry` es el resultado de la última ronda de iteración.
- `$ item` es el valor de la posición actual en la matriz.

Suma de la matriz

```
$result = array_reduce([1, 2, 3, 4, 5], function($carry, $item) {
    return $carry + $item;
});
```

resultado: 15

El número más grande en la matriz

```
$result = array_reduce([10, 23, 211, 34, 25], function($carry, $item) {
    return $item > $carry ? $item : $carry;
});
```

resultado: 211

Es todo el artículo más de 100

```
$result = array_reduce([101, 230, 210, 341, 251], function($carry, $item) {
    return $carry && $item > 100;
}, true); //default value must set true
```

resultado: true

Es cualquier artículo menos de 100

```
$result = array_reduce([101, 230, 21, 341, 251], function($carry, $item) {
    return $carry || $item < 100;
}, false); //default value must set false
```

resultado: true

Como implosionar (\$ array, \$ pieza)

```
$result = array_reduce(["hello", "world", "PHP", "language"], function($carry, $item) {
    return !$carry ? $item : $carry . "-" . $item ;
});
```

resultado: "hello-world-PHP-language"

Si se hace un método de implosión, el código fuente será:

```

function implode_method($array, $piece) {
    return array_reduce($array, function($carry, $item) use ($piece) {
        return !$carry ? $item : ($carry . $piece . $item);
    });
}

$result = implode_method(["hello", "world", "PHP", "language"], "-");

```

resultado: "hello-world-PHP-language"

Arreglos de "destrucción" usando list ()

Use [list \(\)](#) para asignar rápidamente una lista de valores variables a una matriz. Ver también [compacta \(\)](#)

```

// Assigns to $a, $b and $c the values of their respective array elements in $array
// with keys numbered from zero
list($a, $b, $c) = $array;

```

Con PHP 7.1 (actualmente en versión beta) podrá usar la [sintaxis de la lista corta](#) :

```

// Assigns to $a, $b and $c the values of their respective array elements in $array with keys
// numbered from zero
[$a, $b, $c] = $array;

// Assigns to $a, $b and $c the values of the array elements in $array with the keys "a", "b"
// and "c", respectively
["a" => $a, "b" => $b, "c" => $c] = $array;

```

Presionar un valor en una matriz

Hay dos formas de insertar un elemento en una matriz: `array_push` y `$array[] =`

El `array_push` se usa así:

```

$array = [1,2,3];
$newArraySize = array_push($array, 5, 6); // The method returns the new size of the array
print_r($array); // Array is passed by reference, therefore the original array is modified to
contain the new elements

```

Este código se imprimirá:

```

Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)

```

`$array[] = se usa así:`

```
$array = [1,2,3];
$array[] = 5;
$array[] = 6;
print_r($array);
```

Este código se imprimirá:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

Lea Ejecutando sobre una matriz en línea: <https://riptutorial.com/es/php/topic/6826/ejecutando-sobre-una-matriz>

Capítulo 36: Enchufes

Examples

Socket cliente TCP

Creando un socket que usa el TCP (Protocolo de Control de Transmisión)

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Asegúrese de que el zócalo se haya creado correctamente. La función `onSocketFailure` proviene del ejemplo [Manejo de errores de socket](#) en este tema.

```
if(!is_resource($socket)) onSocketFailure("Failed to create socket");
```

Conecte el zócalo a una dirección especificada

La segunda línea falla con gracia si falla la conexión.

```
socket_connect($socket, "chat.stackoverflow.com", 6667)  
or onSocketFailure("Failed to connect to chat.stackoverflow.com:6667", $socket);
```

Enviando datos al servidor

La función `socket_write` envía bytes a través de un socket. En PHP, una matriz de bytes está representada por una cadena, que normalmente es insensible a la codificación.

```
socket_write($socket, "NICK Alice\r\nUSER alice 0 * :Alice\r\n");
```

Recibiendo datos del servidor

El siguiente fragmento de `socket_read` recibe algunos datos del servidor mediante la función `socket_read`.

Al pasar `PHP_NORMAL_READ` como el tercer parámetro se lee hasta un byte `\r / \n`, y este byte se

incluye en el valor de retorno.

Al pasar `PHP_BINARY_READ`, por el contrario, lee la cantidad requerida de datos de la secuencia.

Si `socket_set_nonblock` fue llamado antes, y `PHP_BINARY_READ` se utiliza, `socket_read` volverá `false` inmediatamente. De lo contrario, el método se bloquea hasta que se reciben suficientes datos (para alcanzar la longitud en el segundo parámetro, o para alcanzar un final de línea), o se cierra el zócalo.

Este ejemplo lee datos de un servidor supuestamente IRC.

```
while(true) {
    // read a line from the socket
    $line = socket_read($socket, 1024, PHP_NORMAL_READ);
    if(substr($line, -1) === "\r") {
        // read/skip one byte from the socket
        // we assume that the next byte in the stream must be a \n.
        // this is actually bad in practice; the script is vulnerable to unexpected values
        socket_read($socket, 1, PHP_BINARY_READ);
    }

    $message = parseLine($line);
    if($message->type === "QUIT") break;
}
```

Cerrando el zócalo

Cerrar el socket libera el socket y sus recursos asociados.

```
socket_close($socket);
```

Zócalo del servidor TCP

Creación de zócalo

Crea un socket que use el TCP. Es lo mismo que crear un socket de cliente.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Enlace de zócalo

Enlace las conexiones de una red dada (parámetro 2) para un puerto específico (parámetro 3) al socket.

El segundo parámetro suele ser "`0.0.0.0`", que acepta la conexión de todas las redes. También puede

Una causa común de errores de `socket_bind` es que la dirección especificada ya está vinculada a otro proceso . Otros procesos generalmente se eliminan (generalmente de forma manual para evitar la muerte accidental de procesos críticos) para que los sockets se liberen.

```
socket_bind($socket, "0.0.0.0", 6667) or onSocketFailure("Failed to bind to 0.0.0.0:6667");
```

Establecer un zócalo para escuchar.

Haz que el socket escuche las conexiones entrantes usando `socket_listen` . El segundo parámetro es el número máximo de conexiones para permitir la puesta en cola antes de que sean aceptadas.

```
socket_listen($socket, 5);
```

Manejo de conexión

Un servidor TCP es en realidad un servidor que maneja conexiones secundarias. `socket_accept` crea una nueva conexión secundaria.

```
$conn = socket_accept($socket);
```

La transferencia de datos para una conexión desde `socket_accept` es la misma que para un socket de cliente TCP .

Cuando se debe cerrar esta conexión, llame a `socket_close($conn)` ; directamente. Esto no afectará el socket del servidor TCP original.

Cerrando el servidor

Por otro lado, `socket_close($socket)` ; debe llamarse cuando el servidor ya no se utiliza. Esto también liberará la dirección TCP, permitiendo que otros procesos se unan a la dirección.

Manejo de errores de socket

`socket_last_error` se puede usar para obtener el ID de error del último error de la extensión de sockets.

`socket_strerror` se puede usar para convertir el ID en cadenas legibles para el usuario.

```
function onSocketFailure(string $message, $socket = null) {
    if(is_resource($socket)) {
        $message .= ": " . socket_strerror(socket_last_error($socket));
    }
    die($message);
```

```
}
```

Servidor UDP socket

Un servidor UDP (protocolo de datagramas de usuario), a diferencia de TCP, no está basado en flujo. Se basa en paquetes, es decir, un cliente envía datos en unidades denominadas "paquetes" al servidor, y el cliente identifica a los clientes por su dirección. No hay una función incorporada que relacione diferentes paquetes enviados desde el mismo cliente (a diferencia de TCP, donde los datos del mismo cliente son manejados por un recurso específico creado por `socket_accept`). Puede pensarse como una nueva conexión TCP es aceptada y cerrada cada vez que llega un paquete UDP.

Creando un socket de servidor UDP

```
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
```

Atar un socket a una dirección

Los parámetros son los mismos que para un servidor TCP.

```
socket_bind($socket, "0.0.0.0", 9000) or onSocketFailure("Failed to bind to 0.0.0.0:9000", $socket);
```

Enviando un paquete

Esta línea envía `$data` en un paquete UDP a `$address : $port`.

```
socket_sendto($socket, $data, strlen($data), 0, $address, $port);
```

Recibiendo un paquete

El siguiente fragmento de código intenta administrar paquetes UDP de una manera indexada por el cliente.

```
$clients = [];
while (true){
    socket_recvfrom($socket, $buffer, 32768, 0, $ip, $port) === true
        or onSocketFailure("Failed to receive packet", $socket);
    $address = "$ip:$port";
    if (!isset($clients[$address])) $clients[$address] = new Client();
    $clients[$address]->handlePacket($buffer);
}
```

Cerrando el servidor

`socket_close` se puede utilizar en el recurso de socket del servidor UDP. Esto liberará la dirección UDP, permitiendo que otros procesos se unan a esta dirección.

Lea Enchufes en línea: <https://riptutorial.com/es/php/topic/6138/enchufes>

Capítulo 37: Enviando email

Parámetros

Parámetro	Detalles
string \$to	La dirección de correo electrónico del destinatario
string \$subject	La linea de asunto
string \$message	El cuerpo del email.
string \$additional_headers	Opcional: encabezados para agregar al correo electrónico.
string \$additional_parameters	Opcional: argumentos para pasar a la aplicación de envío de correo configurada en la línea de comandos

Observaciones

Correo electrónico que estoy enviando a través de mi script nunca llega. ¿Qué tengo que hacer?

- Asegúrese de tener activado el informe de errores para ver cualquier error.
- Si tiene acceso a los archivos de registro de errores de PHP, verifique esos.
- ¿El comando `mail()` configurado correctamente en su servidor? (Si estás en un alojamiento compartido, no puedes cambiar nada aquí).
- Si los correos electrónicos están desapareciendo, inicie una cuenta de correo electrónico con un servicio de correo electrónico gratuito que tenga una carpeta de correo no deseado (o use una cuenta de correo electrónico que no filtre el correo basura). De esta manera, puede ver si el correo electrónico no se envía, o tal vez se envía, pero se filtra como correo no deseado.
- ¿Revisó la dirección "de:" que utilizó para los posibles correos "devueltos al remitente"? También puede configurar una dirección de rebote separada para los correos de error.

El correo electrónico que estoy enviando se está filtrando como spam. ¿Qué tengo que hacer?

- ¿La dirección del remitente ("De") pertenece a un dominio que se ejecuta en el servidor desde el que envía el correo electrónico? Si no, cambia eso.

Nunca use direcciones de remitentes como `xxx@gmail.com`. Utilice `reply-to` si necesita respuestas para llegar a una dirección diferente.

- ¿Está su servidor en una lista negra? Esta es una posibilidad cuando estás en un alojamiento compartido cuando los vecinos se comportan mal. La mayoría de los proveedores de listas negras, como [Spamhaus](#), tienen herramientas que le permiten buscar la IP de su servidor. También hay herramientas de terceros como [MX Toolbox](#).
- Algunas instalaciones de PHP requieren establecer un [quinto parámetro](#) en mail () para agregar una dirección de remitente. Ver si este podría ser el caso para usted.
- Si todo lo demás falla, considere usar el correo electrónico como un servicio como [Mailgun](#), [SparkPost](#), [Amazon SES](#), [Mailjet](#), [SendinBlue](#) o [SendGrid](#), para nombrar algunos, en su lugar. Todos ellos tienen API que se pueden llamar usando PHP.

Examples

Envío de correo electrónico: conceptos básicos, más detalles y un ejemplo completo

Un correo electrónico típico tiene tres componentes principales:

1. Un destinatario (representado como una dirección de correo electrónico)
2. Un sujeto
3. Un cuerpo de mensaje

Enviar correo en PHP puede ser tan simple como llamar a la función incorporada `mail()`. `mail()` toma hasta cinco parámetros, pero los tres primeros son todo lo que se requiere para enviar un correo electrónico (aunque los cuatro parámetros se usan comúnmente como se demostrará a continuación). Los tres primeros parámetros son:

1. La dirección de correo electrónico del destinatario (cadena)
2. El asunto del correo electrónico (cadena)
3. El cuerpo del correo electrónico (cadena) (por ejemplo, el contenido del correo electrónico)

Un ejemplo mínimo se asemejaría al siguiente código:

```
mail('recipient@example.com', 'Email Subject', 'This is the email message body');
```

El ejemplo simple anterior funciona bien en circunstancias limitadas, como codificar una alerta de correo electrónico para un sistema interno. Sin embargo, es común colocar los datos pasados como parámetros para `mail()` en variables para hacer que el código sea más limpio y fácil de administrar (por ejemplo, generar dinámicamente un correo electrónico desde un envío de formulario).

Además, `mail()` acepta un cuarto parámetro que le permite recibir encabezados de correo adicionales con su correo electrónico. Estos encabezados pueden permitirle establecer:

- la `From` nombre y dirección de correo electrónico que el usuario verá
- el `Reply-To` dirección de correo electrónico se enviará la respuesta del usuario a
- encabezados adicionales no estándares como `X-Mailer` que pueden indicar al destinatario

que este correo electrónico se envió a través de PHP

```
$to      = 'recipient@example.com';           // Could also be $to      =
$_POST['recipient'];
$subject = 'Email Subject';                   // Could also be $subject = $_POST['subject'];

$message = 'This is the email message body'; // Could also be $message = $_POST['message'];

$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);

```

El quinto parámetro opcional se puede usar para pasar banderas adicionales como opciones de línea de comando al programa configurado para ser usado al enviar correo, según lo definido por la configuración de `sendmail_path`. Por ejemplo, esto se puede usar para configurar la dirección del remitente del sobre cuando se usa `sendmail` / `postfix` con la opción `-f sendmail`.

```
$fifth = '-fno-reply@example.com';
```

Aunque el uso de `mail()` puede ser bastante confiable, de ninguna manera se garantiza que se enviará un correo `mail()` cuando se llame a `mail()`. Para ver si hay un error potencial al enviar su correo electrónico, debe capturar el valor de retorno de `mail()`. `TRUE` será devuelto si el correo fue aceptado exitosamente para su entrega. De lo contrario, recibirá `FALSE`.

```
$result = mail($to, $subject, $message, $headers, $fifth);
```

NOTA : aunque `mail()` puede devolver `TRUE`, *no* significa que el correo electrónico fue enviado o que el destinatario lo recibirá. Solo indica que el correo se entregó con éxito al sistema de correo de su sistema con éxito.

Si desea enviar un correo electrónico HTML, no hay mucho trabajo que deba hacer. Necesitas:

1. Añadir el encabezado de la `MIME-Version`
2. Añadir el encabezado de `Content-Type`
3. Asegúrese de que su contenido de correo electrónico es HTML

```
$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = '<html><body>This is the email message body</body></html>';
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'MIME-Version: 1.0',
    'Content-Type: text/html; charset=ISO-8859-1',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Aquí hay un ejemplo completo de cómo usar la función `mail()` PHP `mail()`

```

<?php

// Debugging tools. Only turn these on in your development environment.

error_reporting(-1);
ini_set('display_errors', 'On');
set_error_handler("var_dump");

// Special mail settings that can make mail less likely to be considered spam
// and offers logging in case of technical difficulties.

ini_set("mail.log", "/tmp/mail.log");
ini_set("mail.add_x_header", TRUE);

// The components of our email

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';
$headers = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
}

// Send the email

$result = mail($to, $subject, $message, $headers);

// Check the results and react accordingly

if ($result) {

    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;

}
else {

    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.

}

```

Ver también

Documentación oficial

- [mail\(\)](#)
- [Configuración de mail\(\) PHP mail\(\)](#)

Preguntas relacionadas sobre el desbordamiento de pila

- [Formulario de correo PHP no completa el envío de correo electrónico](#)
- [¿Cómo se asegura de que el correo electrónico que envíe programáticamente no se marque](#)

- automáticamente como spam?
- Cómo usar SMTP para enviar correos electrónicos
- Configuración del sobre de la dirección

Mailers alternativos

- [PHPMailer](#)
- [SwiftMailer](#)
- [PERA :: Correo](#)

Servidores de correo electrónico

- [Mercury Mail \(Windows\)](#)

Temas relacionados

- [Publicar / Redirigir / Obtener](#)

Enviando correo electrónico HTML usando `correo()`

```
<?php
$to      = 'recipient@example.com';
$subject = 'Sending an HTML email using mail() in PHP';
$message = '<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is italic.</i></p></body></html>';

$headers = implode("\r\n", [
    "From: John Conde <webmaster@example.com>",
    "Reply-To: webmaster@example.com",
    "X-Mailer: PHP/" . PHP_VERSION,
    "MIME-Version: 1.0",
    "Content-Type: text/html; charset=UTF-8"
]);
mail($to, $subject, $message, $headers);
```

Esto no es muy diferente a [enviar un correo electrónico de texto sin formato](#). Las diferencias clave en cuanto al cuerpo del contenido están estructuradas como un documento HTML y hay dos encabezados adicionales que deben incluirse para que el cliente de correo electrónico sepa que el correo electrónico es renderizado como HTML. Son:

- Versión MIME: 1.0
- Tipo de contenido: texto / html; conjunto de caracteres = UTF-8

Enviar correo electrónico de texto sin formato utilizando PHPMailer

Email basico de texto

```
<?php
$mail = new PHPMailer();
```

```

$mail->From      = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject   = "Subject Text";
$mail->Body       = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Mailer Error: " . $mail->ErrorInfo;
}

```

Adición de destinatarios adicionales, destinatarios CC, destinatarios BCC

```

<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recipient1@example.com", "Recipient Name");
$mail->addAddress("recipient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject   = "Subject Text";
$mail->Body       = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}

```

Enviar correo electrónico con un archivo adjunto utilizando correo ()

```

<?php

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';

$attachment = '/path/to/your/file.pdf';
$content = file_get_contents($attachment);

/* Attachment content transferred in Base64 encoding

```

```

MUST be split into chunks 76 characters in length as
specified by RFC 2045 section 6.8. By default, the
function chunk_split() uses a chunk length of 76 with
a trailing CRLF (\r\n). The 76 character requirement
does not include the carriage return and line feed */
$content = chunk_split(base64_encode($content));

/* Boundaries delimit multipart entities. As stated
in RFC 2046 section 5.1, the boundary MUST NOT occur
in any encapsulated part. Therefore, it should be
unique. As stated in the following section 5.1.1, a
boundary is defined as a line consisting of two hyphens
("--"), a parameter value, optional linear whitespace,
and a terminating CRLF. */
$prefix      = "part_"; // This is an optional prefix
/* Generate a unique boundary parameter value with our
prefix using the uniqid() function. The second parameter
makes the parameter value more unique. */
$boundary    = uniqid($prefix, true);

// headers
$headers     = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION,
    'MIME-Version: 1.0',
    // boundary parameter required, must be enclosed by quotes
    'Content-Type: multipart/mixed; boundary=' . $boundary . "'",
    "Content-Transfer-Encoding: 7bit",
    "This is a MIME encoded message." // message for restricted transports
]);

// message and attachment
$message     = implode("\r\n", [
    "--" . $boundary, // header boundary delimiter line
    'Content-Type: text/plain; charset="iso-8859-1"',
    "Content-Transfer-Encoding: 8bit",
    $message,
    "--" . $boundary, // content boundary delimiter line
    'Content-Type: application/octet-stream; name="RenamedFile.pdf"',
    "Content-Transfer-Encoding: base64",
    "Content-Disposition: attachment",
    $content,
    "--" . $boundary . "--" // closing boundary delimiter line
]);
}

$result = mail($to, $subject, $message, $headers); // send the email

if ($result) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.
}

```

Codificaciones de transferencia de contenido

Las codificaciones disponibles son `7bit`, `8bit`, `binary`, `quoted-printable`, `base64`, `ietf-token` y `x-token`. De estas codificaciones, cuando un encabezado tiene un Tipo de contenido de *varias partes*, la Codificación de transferencia de contenido **no debe** tener ningún otro valor que no sea `7bit`, `8bit` o `binario` como se indica en RFC 2045, sección 6.4.

Nuestro ejemplo elige la codificación de 7 bits, que representa los caracteres US-ASCII, para el encabezado de varias partes porque, como se señala en la sección 6 de RFC 2045, algunos protocolos solo admiten esta codificación. Los datos dentro de los límites se pueden codificar parte por parte (RFC 2046, sección 5.1). Este ejemplo hace exactamente esto. La primera parte, que contiene el mensaje de texto / sin formato, se define como 8 bits, ya que puede ser necesario admitir caracteres adicionales. En este caso, se está utilizando el conjunto de caracteres Latin1 (iso-8859-1). La segunda parte es el archivo adjunto y, por lo tanto, se define como una aplicación codificada en base64 / flujo de octetos. Como base64 transforma datos arbitrarios en el rango de 7 bits, puede enviarse en transportes restringidos (RFC 2045, sección 6.2).

Envío de correo electrónico HTML utilizando PHPMailer

```
<?php

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recipient1@example.com", "Recipient Name");
$mail->addAddress("recipient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject   = "Subject Text";
$mail->isHTML(true);
$mail->Body      = "<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is italic.</i></p></body></html>";
$mail->AltBody = "This paragraph is not bold.\n\nThis text is not italic.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Envío de correo electrónico con un archivo adjunto utilizando PHPMailer

```
<?php
```

```

$mail = new PHPMailer();

$mail->From      = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject   = "Subject Text";
$mail->Body       = "This is a sample basic text email with an attachment using PHPMailer.';

// Add Static Attachment
$attachment = '/path/to/your/file.pdf';
$mail->AddAttachment($attachment , 'RenamedFile.pdf');

// Add Second Attachment, run-time created. ie: CSV to be open with Excel
$csvHeader = "header1,header2,header3";
$csvData = "row1col1,row1col2,row1col3\nrow2col1,row2col2,row2col3";

$mail->AddStringAttachment($csvHeader ."\n" . $csvData, 'your-csv-file.csv', 'base64',
'application/vnd.ms-excel');

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}

```

Enviar correo electrónico de texto sin formato utilizando Sendgrid

Email basico de texto

```

<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
    ->setFrom("sender@example.com")
    ->setSubject("Subject Text")
    ->setText("This is a sample basic text email using ");

$sendgrid->send($email);

```

Adición de destinatarios adicionales, destinatarios CC, destinatarios BCC

```

<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
    ->setFrom("sender@example.com")
    ->setSubject("Subject Text")
    ->setHtml("<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is

```

```

italic.</i></p></body></html>");

$personalization = new Personalization();
$email = new Email("Recipient Name", "recipient1@example.com");
$personalization->addTo($email);
$email = new Email("RecipientCC Name", "recipient2@example.com");
$personalization->addCc($email);
$email = new Email("RecipientBCC Name", "recipient3@example.com");
$personalization->addBcc($email);
$email->addPersonalization($personalization);

$sendgrid->send($email);

```

Enviar correo electrónico con un archivo adjunto utilizando Sendgrid

```

<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
    ->setFrom("sender@example.com")
    ->setSubject("Subject Text")
    ->setText("This is a sample basic text email using ");

$attachment = '/path/to/your/file.pdf';
$content    = file_get_contents($attachment);
$content    = chunk_split(base64_encode($content));

$attachment = new Attachment();
$attachment->setContent($content);
$attachment->setType("application/pdf");
$attachment->setFilename("RenamedFile.pdf");
$attachment->setDisposition("attachment");
$email->addAttachment($attachment);

$sendgrid->send($email);

```

Lea Enviando email en línea: <https://riptutorial.com/es/php/topic/458/enviando-email>

Capítulo 38: Errores comunes

Examples

\$ Inesperado fin

```
Parse error: syntax error, unexpected end of file in C:\xampp\htdocs\stack\index.php on line 4
```

Si recibe un error como este (o, a veces, el `unexpected $end`, según la versión de PHP), deberá asegurarse de que haya hecho coincidir todas las comas invertidas, todos los paréntesis, todas las llaves, todos los corchetes, etc.

El siguiente código produjo el error anterior:

```
<?php  
if (true) {  
    echo "asdf";  
?>
```

Fíjate en la llave que falta. También tenga en cuenta que el número de línea que se muestra para este error es irrelevante, siempre muestra la última línea de su documento.

Llame a `fetch_assoc` en boolean

Si recibe un error como este:

```
Fatal error: Call to a member function fetch_assoc() on boolean in  
C:\xampp\htdocs\stack\index.php on line 7
```

Otras variaciones incluyen algo a lo largo de las líneas de:

```
mysql_fetch_assoc() expects parameter 1 to be resource, boolean given...
```

Estos errores significan que hay algo mal con su consulta (esto es un error de PHP / MySQL), o sus referencias. El error anterior fue producido por el siguiente código:

```
$mysqli = new mysqli("localhost", "root", "");  
  
$query = "SELCT * FROM db"; // notice the errors here  
$result = $mysqli->query($query);  
  
$row = $result->fetch_assoc();
```

Para "arreglar" este error, se recomienda hacer excepciones de lanzamiento de mysql en su lugar:

```
// add this at the start of the script
```

```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

Esto lanzará una excepción con este mensaje mucho más útil en su lugar:

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'SELCT * FROM db' at line 1
```

Otro ejemplo que produciría un error similar, es donde simplemente le dio la información incorrecta a la función `mysql_fetch_assoc` o similar:

```
$john = true;  
mysql_fetch_assoc($john, $mysqli); // this makes no sense??
```

Lea Errores comunes en línea: <https://riptutorial.com/es/php/topic/3830/errores-comunes>

Capítulo 39: Espacios de nombres

Observaciones

De la [documentación de PHP](#) :

¿Qué son los espacios de nombres? En la definición más amplia, los espacios de nombres son una forma de encapsular elementos. Esto puede verse como un concepto abstracto en muchos lugares. Por ejemplo, en cualquier sistema operativo, los directorios sirven para agrupar archivos relacionados y actúan como un espacio de nombres para los archivos que contienen. Como ejemplo concreto, el archivo foo.txt puede existir tanto en el directorio / home / greg como en / home / other, pero dos copias de foo.txt no pueden coexistir en el mismo directorio. Además, para acceder al archivo foo.txt fuera del directorio / home / greg, debemos añadir el nombre del directorio al nombre del archivo usando el separador de directorios para obtener /home/greg/foo.txt. Este mismo principio se extiende a los espacios de nombres en el mundo de la programación.

Tenga en cuenta que los espacios de nombres de nivel superior `PHP` y `php` están reservados para el propio lenguaje PHP. No deben utilizarse en ningún código personalizado.

Examples

Declarando espacios de nombres

Una declaración de espacio de nombres puede verse como sigue:

- `namespace MyProject;` - Declarar el espacio de nombres `MyProject`
- `namespace MyProject\Security\Cryptography;` - Declarar un espacio de nombres anidado
- `namespace MyProject { ... };` : declara un espacio de nombres entre paréntesis.

Se recomienda declarar solo un único espacio de nombres por archivo, aunque puede declarar tantos como desee en un solo archivo:

```
namespace First {
    class A { ... }; // Define class A in the namespace First.
}

namespace Second {
    class B { ... }; // Define class B in the namespace Second.
}

namespace {
    class C { ... }; // Define class C in the root namespace.
}
```

Cada vez que declare un espacio de nombres, las clases que defina después pertenecerán a ese espacio de nombres:

```
namespace MyProject\Shapes;

class Rectangle { ... }
class Square { ... }
class Circle { ... }
```

Una declaración de espacio de nombres se puede utilizar varias veces en diferentes archivos. El ejemplo anterior definió tres clases en el espacio de nombres `MyProject\Shapes` en un solo archivo. Preferiblemente, esto se dividiría en tres archivos, cada uno comenzando con el `namespace MyProject\Shapes;`. Esto se explica con más detalle en el ejemplo estándar de PSR-4.

Hacer referencia a una clase o función en un espacio de nombres

Como se muestra en [Declaración de espacios de nombres](#), podemos definir una clase en un espacio de nombres de la siguiente manera:

```
namespace MyProject\Shapes;

class Rectangle { ... }
```

Para hacer referencia a esta clase, se debe utilizar la ruta completa (incluido el espacio de nombres):

```
$rectangle = new MyProject\Shapes\Rectangle();
```

Esto se puede reducir importando la clase a través de la declaración de `use`:

```
// Rectangle becomes an alias to MyProject\Shapes\Rectangle
use MyProject\Shapes\Rectangle;

$rectangle = new Rectangle();
```

En cuanto a PHP 7.0, puede agrupar varias declaraciones de `use` en una sola declaración utilizando corchetes:

```
use MyProject\Shapes\{
    Rectangle,           //Same as `use MyProject\Shapes\Rectangle`
    Circle,             //Same as `use MyProject\Shapes\Circle`
    Triangle,           //Same as `use MyProject\Shapes\Triangle`

    Polygon\FiveSides, //You can also import sub-namespaces
    Polygon\SixSides   //In a grouped `use`-statement
};

$rectangle = new Rectangle();
```

A veces dos clases tienen el mismo nombre. Esto no es un problema si se encuentran en un espacio de nombres diferente, pero podría convertirse en un problema al intentar importarlos con la declaración de `use`:

```
use MyProject\Shapes\Oval;
```

```
use MyProject\Languages\Oval; // Apparently Oval is also a language!
// Error!
```

Esto se puede resolver definiendo un nombre para el alias usando la palabra clave `as`:

```
use MyProject\Shapes\Oval as OvalShape;
use MyProject\Languages\Oval as OvalLanguage;
```

Para hacer referencia a una clase fuera del espacio de nombres actual, debe escaparse con una `\`, de lo contrario se asume una ruta de espacio de nombres relativa a partir del espacio de nombres actual:

```
namespace MyProject\Shapes;

// References MyProject\Shapes\Rectangle. Correct!
$a = new Rectangle();

// References MyProject\Shapes\Rectangle. Correct, but unneeded!
$a = new \MyProject\Shapes\Rectangle();

// References MyProject\Shapes\MyProject\Shapes\Rectangle. Incorrect!
$a = new MyProject\Shapes\Rectangle();

// Referencing StdClass from within a namespace requires a \ prefix
// since it is not defined in a namespace, meaning it is global.

// References StdClass. Correct!
$a = new \StdClass();

// References MyProject\Shapes\StdClass. Incorrect!
$a = new StdClass();
```

¿Qué son los espacios de nombres?

La comunidad de PHP tiene muchos desarrolladores que crean muchos códigos. Esto significa que el código PHP de una biblioteca puede usar el mismo nombre de clase que otra biblioteca. Cuando ambas bibliotecas se usan en el mismo espacio de nombres, chocan y causan problemas.

Los espacios de nombres resuelven este problema. Como se describe en el manual de referencia de PHP, los espacios de nombres se pueden comparar con los directorios del sistema operativo que contienen los archivos de espacios de nombres; dos archivos con el mismo nombre pueden coexistir en directorios separados. Del mismo modo, dos clases de PHP con el mismo nombre pueden coexistir en espacios de nombres de PHP separados.

Es importante que escriba un espacio de nombre en su código para que otros desarrolladores puedan utilizarlo sin temor a colisionar con otras bibliotecas.

Declarar sub-espacios de nombres

Para declarar un solo espacio de nombres con jerarquía, utilice el siguiente ejemplo:

```
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */ }
```

El ejemplo anterior crea:

constante MyProject\Sub\Level\CONNECT_OK

clase MyProject\Sub\Level\Connection y

función MyProject\Sub\Level\connect

Lea Espacios de nombres en línea: <https://riptutorial.com/es/php/topic/1021/espacios-de-nombres>

Capítulo 40: Estructuras de Control

Examples

Sintaxis alternativa para estructuras de control.

PHP proporciona una sintaxis alternativa para algunas estructuras de control: `if`, `while`, `for`, `foreach` y `switch`.

Cuando se compara con la sintaxis normal, la diferencia es, que la llave de apertura se sustituye por dos puntos (`:`) y la llave de cierre se sustituye por `endif`; al `endwhile`; `endfor`; , `endforeach`; , o `endswitch`; , respectivamente. Para ejemplos individuales, vea el tema sobre [la sintaxis alternativa para las estructuras de control](#).

```
if ($a == 42):
    echo "The answer to life, the universe and everything is 42.";
endif;
```

Múltiples declaraciones de `elseif` que usan sintaxis corta:

```
if ($a == 5):
    echo "a equals 5";
elseif ($a == 6):
    echo "a equals 6";
else:
    echo "a is neither 5 nor 6";
endif;
```

[Manual de PHP - Estructuras de control - Sintaxis alternativa](#)

mientras

`while` bucle se repite a través de un bloque de código siempre que una condición especificada sea verdadera.

```
$i = 1;
while ($i < 10) {
    echo $i;
    $i++;
}
```

Salida: 123456789

Para obtener información detallada, consulte [el tema Bucles](#).

hacer mientras

`do-while` bucle `do-while` `while` primero ejecuta un bloque de código una vez, en cada caso, luego

itera a través de ese bloque de código siempre que una condición específica sea verdadera.

```
$i = 0;
do {
    $i++;
    echo $i;
} while ($i < 10);

Output: `12345678910`
```

Para obtener información detallada, consulte [el tema Bucles](#).

ir

El operador `goto` permite saltar a otra sección en el programa. Está disponible desde PHP 5.3.

La instrucción `goto` es un `goto` seguido de la etiqueta de destino deseada: `goto MyLabel;`.

El objetivo del salto se especifica mediante una etiqueta seguida de dos puntos: `MyLabel:`

Este ejemplo imprimirá `Hello World!`:

```
<?php
goto MyLabel;
echo 'This text will be skipped, because of the jump.';

MyLabel:
echo 'Hello World!';
?>
```

declarar

`declare` se utiliza para establecer una directiva de ejecución para un bloque de código.

Se reconocen las siguientes directivas:

- `ticks`
- `encoding`
- `strict_types`

Por ejemplo, establezca marcas en 1:

```
declare(ticks=1);
```

Para habilitar el modo de tipo estricto, la `declare` declaración se utiliza con la declaración `strict_types`:

```
declare(strict_types=1);
```

si mas

La instrucción `if` en el ejemplo anterior permite ejecutar un fragmento de código, cuando se cumple la condición. Cuando desea ejecutar un fragmento de código, cuando no se cumple la condición, puede extender el `if` con `else`.

```
if ($a > $b) {  
    echo "a is greater than b";  
} else {  
    echo "a is NOT greater than b";  
}
```

Manual de PHP - Estructuras de control - Si no

El operador ternario como sintaxis abreviada para if-else

El [operador ternario](#) evalúa algo basándose en que una condición es verdadera o no. Es un operador de comparación y se usa a menudo para expresar una condición simple en caso de que exista en una forma más corta. Permite probar rápidamente una condición y, a menudo, reemplaza una línea multilínea `if`, lo que hace que su código sea más compacto.

Este es el ejemplo anterior utilizando una expresión ternaria y valores de variables: `$a=1; $b=2;`

```
echo ($a > $b) ? "a is greater than b" : "a is NOT greater than b";
```

Salidas: `a is NOT greater than b`.

incluir y requerir

exigir

`require` es similar a `include`, excepto que producirá un error de nivel `E_COMPILE_ERROR` fatal en `E_COMPILE_ERROR` error. Cuando el `require` falla, detendrá el script. Cuando la `include` falla, no detendrá la secuencia de comandos y solo emitirá `E_WARNING`.

```
require 'file.php';
```

Manual de PHP - Estructuras de control - Requerir

incluir

La declaración de `include` incluye y evalúa un archivo.

```
./variables.php
```

```
$a = 'Hello World!';
```

```
. / main.php'
```

```
include 'variables.php';
echo $a;
// Output: `Hello World!`
```

Tenga cuidado con este enfoque, ya que se considera un [olor de código](#), ya que el archivo incluido está modificando la cantidad y el contenido de las variables definidas en el alcance dado.

También puede `include` archivo, que devuelve un valor. Esto es extremadamente útil para manejar matrices de configuración:

`configuracion.php`

```
<?php
return [
    'dbname' => 'my db',
    'user' => 'admin',
    'pass' => 'password',
];
```

`main.php`

```
<?php
$config = include 'configuration.php';
```

Este enfoque evitará que el archivo incluido contamine su alcance actual con variables modificadas o agregadas.

[Manual de PHP - Estructuras de control - Incluir](#)

include & require también se puede usar para asignar valores a una variable cuando se devuelve algo por archivo.

Ejemplo:

archivo `include1.php`:

```
<?php
$a = "This is to be returned";

return $a;
?>
```

archivo `index.php`:

```
$value = include 'include1.php';
// Here, $value = "This is to be returned"
```

[regreso](#)

La declaración de `return` devuelve el control del programa a la función de llamada.

Cuando se llama a `return` desde dentro de una función, la ejecución de la función actual terminará.

```
function returnEndsFunctions()
{
    echo 'This is executed';
    return;
    echo 'This is not executed.';
}
```

Cuando ejecutas `returnEndsFunctions();` obtendrás la salida `This is executed;`

Cuando se llama a `return` dentro de una función con un argumento, la ejecución de la función actual finalizará y el valor del argumento se devolverá a la función que llama.

para

`for` bucles se utilizan normalmente cuando tienes un fragmento de código que deseas repetir un número determinado de veces.

```
for ($i = 1; $i < 10; $i++) {
    echo $i;
}
```

Salidas: 123456789

Para obtener información detallada, consulte [el tema Bucles](#).

para cada

`foreach` es una construcción, que le permite iterar sobre matrices y objetos fácilmente.

```
$array = [1, 2, 3];
foreach ($array as $value) {
    echo $value;
}
```

Salidas: 123 .

Para usar el bucle `foreach` con un objeto, debe implementar la interfaz [Iterator](#).

Cuando iteras sobre matrices asociativas:

```
$array = ['color'=>'red'];

foreach($array as $key => $value) {
    echo $key . ':' . $value;
}
```

Salidas: color: red

Para obtener información detallada, consulte [el tema Bucles](#).

si otra cosa más

elseif

`elseif` combina `if` y `if else`. La instrucción `if` se extiende para ejecutar una instrucción diferente en caso de que la expresión original `if` no se cumpla. Pero, la expresión alternativa solo se ejecuta cuando se cumple la expresión condicional `elseif`.

El siguiente código muestra "a es más grande que b", "a es igual a b" o "a es más pequeño que b":

```
if ($a > $b) {  
    echo "a is bigger than b";  
} elseif ($a == $b) {  
    echo "a is equal to b";  
} else {  
    echo "a is smaller than b";  
}
```

Varias declaraciones de elseif

Puede usar varias sentencias `elseif` dentro de la misma sentencia `if`:

```
if ($a == 1) {  
    echo "a is One";  
} elseif ($a == 2) {  
    echo "a is Two";  
} elseif ($a == 3) {  
    echo "a is Three";  
} else {  
    echo "a is not One, not Two nor Three";  
}
```

Si

La construcción `if` permite la ejecución condicional de fragmentos de código.

```
if ($a > $b) {  
    echo "a is bigger than b";  
}
```

[Manual de PHP - Estructuras de control - Si](#)

cambiar

La estructura del `switch` realiza la misma función que una serie de sentencias `if`, pero puede hacer el trabajo en menos líneas de código. El valor que se va a probar, como se define en la

declaración de `switch`, se compara para igualar con los valores en cada una de las declaraciones de `case` hasta que se encuentra una coincidencia y se ejecuta el código en ese bloque. Si no se encuentra una declaración de `case` coincidente, se ejecuta el código en el bloque `default`, si existe.

Cada bloque de código en un `case` o declaración por `default` debe terminar con la declaración de `break`. Esto detiene la ejecución de la estructura del `switch` y continúa la ejecución del código inmediatamente después. Si se omite la instrucción `break`, se ejecuta el código de la siguiente declaración de `case`, *incluso si no hay coincidencia*. Esto puede causar la ejecución inesperada del código si se olvida la instrucción `break`, pero también puede ser útil cuando varias declaraciones de `case` necesitan compartir el mismo código.

```
switch ($colour) {  
    case "red":  
        echo "the colour is red";  
        break;  
    case "green":  
    case "blue":  
        echo "the colour is green or blue";  
        break;  
    case "yellow":  
        echo "the colour is yellow";  
        // note missing break, the next block will also be executed  
    case "black":  
        echo "the colour is black";  
        break;  
    default:  
        echo "the colour is something else";  
        break;  
}
```

Además de probar valores fijos, la construcción también puede ser obligada a probar declaraciones dinámicas al proporcionar un valor booleano a la instrucción `switch` y cualquier expresión a la declaración del `case`. Tenga en cuenta que se utiliza el *primer* valor coincidente, por lo que el siguiente código dará como resultado "más de 100":

```
$i = 1048;  
switch (true) {  
    case ($i > 0):  
        echo "more than 0";  
        break;  
    case ($i > 100):  
        echo "more than 100";  
        break;  
    case ($i > 1000):  
        echo "more than 1000";  
        break;  
}
```

Para posibles problemas con la escritura suelta mientras se usa la construcción del `switch`, consulte [Cambiar sorpresas](#)

Lea Estructuras de Control en línea: <https://riptutorial.com/es/php/topic/2366/estructuras-de-control>

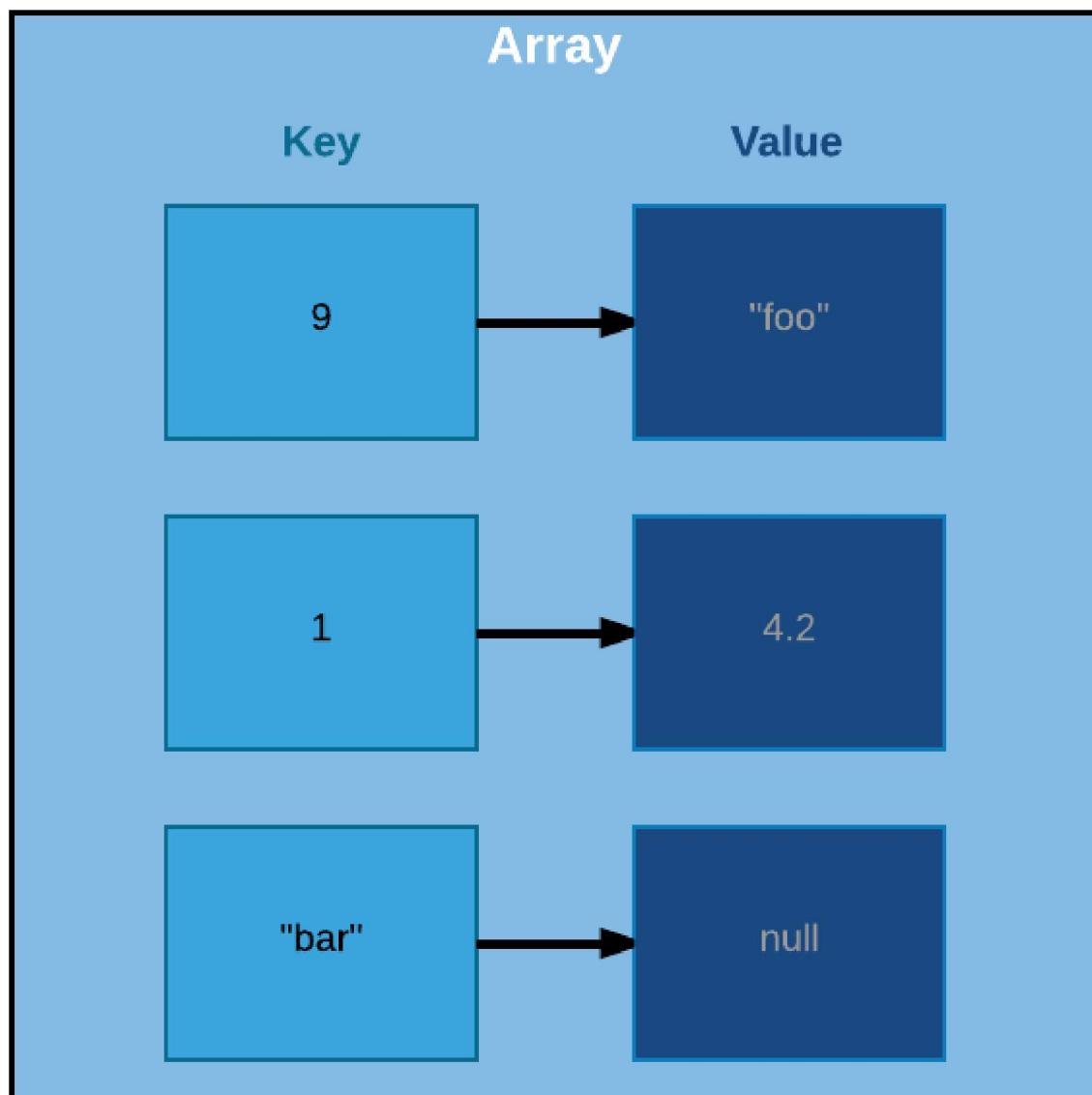
Capítulo 41: Estructuras de datos SPL

Examples

SplFixedArray

Diferencia de PHP Array

El tipo de matriz predeterminado de PHP se implementa realmente como mapas hash ordenados, que nos permiten crear matrices que consisten en pares clave / valor donde los valores pueden ser de cualquier tipo y las claves pueden ser números o cadenas. Sin embargo, esto no es tradicionalmente cómo se crean los arreglos.



Entonces, como puede ver en esta ilustración, una matriz PHP normal se puede ver más como un conjunto ordenado de pares clave / valor, donde cada clave se puede asignar a cualquier valor. Note que en esta matriz tenemos claves que son tanto números como cadenas, como valores de diferentes tipos y la clave no tiene relación con el orden de los elementos.

```
$arr = [
    9      => "foo",
    1      => 4.2,
    "bar"  => null,
];

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

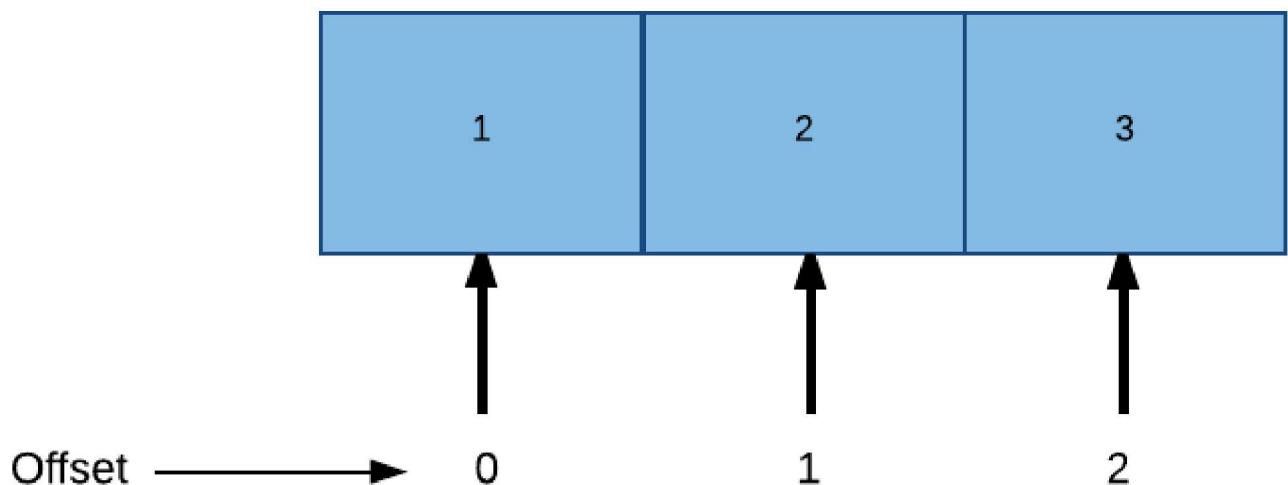
Así que el código anterior nos daría exactamente lo que esperaríamos.

```
9 => foo
1 => 4.2
bar =>
```

Las matrices regulares de PHP también tienen un tamaño dinámico para nosotros. Crecen y se encogen a medida que empujamos y hacemos estallar los valores desde y hacia la matriz, automáticamente.

Sin embargo, en una matriz tradicional, el tamaño es fijo y consiste completamente en el mismo tipo de valor. Además, en lugar de claves, cada valor es el acceso por su índice, que puede deducirse por su desplazamiento en la matriz.

SPLFixedArray



Como sabríamos el tamaño de un tipo dado y el tamaño fijo de la matriz, un desplazamiento es el `type size * n` representa la posición del valor en la matriz. Entonces, en el ejemplo anterior, `$arr[0]` nos da `1`, el primer elemento de la matriz y `$arr[1]` nos da `2`, y así sucesivamente.

`SplFixedArray`, sin embargo, no restringe el tipo de valores. Sólo restringe las claves a los tipos de números. También es de un tamaño fijo.

Esto hace que `SplFixedArrays` sea más eficiente que los arreglos PHP normales de una manera particular. Son más compactos por lo que requieren menos memoria.

Creando la matriz

`SplFixedArray` se implementa como un objeto, pero se puede acceder con la misma sintaxis familiar a la que accede a una matriz PHP normal, ya que implementan la interfaz `ArrayAccess`. También implementan las interfaces `Countable` e `Iterator` para que se comporten de la misma manera que lo harías con las matrices que se comportan en PHP (es decir, cosas como `count($arr)` y `foreach($arr as $k => $v)` funcionan de la misma manera para `SplFixedArray` ya que hacen arreglos normales en PHP).

El constructor `SplFixedArray` toma un argumento, que es el tamaño de la matriz.

```
$arr = new SplFixedArray(4);

$arr[0] = "foo";
$arr[1] = "bar";
$arr[2] = "baz";

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Esto te da lo que esperas.

```
0 => foo
1 => bar
2 => baz
3 =>
```

Esto también funciona como se esperaba.

```
var_dump(count($arr));
```

Nos da...

```
int(4)
```

Note que en `SplFixedArray`, a diferencia de una matriz PHP normal, la clave representa el orden del elemento en nuestra matriz, porque es un *índice verdadero* y no solo un *mapa*.

Cambiar el tamaño de la matriz

Solo tenga en cuenta que debido a que la matriz es de un tamaño fijo, la cuenta siempre devolverá el mismo valor. Así que mientras `unset($arr[1])` dará como resultado `$arr[1] === null`, la `count($arr)` aún permanece en `4`.

Por lo tanto, para cambiar el tamaño de la matriz, deberá llamar al método `setSize`.

```
$arr->setSize(3);

var_dump(count($arr));

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Ahora tenemos ...

```
int(3)
0 => foo
1 =>
2 => baz
```

Importar a SplFixedArray y exportar desde SplFixedArray

También puede importar / exportar una matriz PHP normal dentro y fuera de un `SplFixedArray` con los métodos `fromArray` y `toArray`.

```
$array      = [1,2,3,4,5];
$fixedArray = SplFixedArray::fromArray($array);

foreach($fixedArray as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Yendo por el otro lado.

```
$fixedArray = new SplFixedArray(5);

$fixedArray[0] = 1;
$fixedArray[1] = 2;
```

```
$fixedArray[2] = 3;  
$fixedArray[3] = 4;  
$fixedArray[4] = 5;  
  
$array = $fixedArray->toArray();  
  
foreach($array as $value) {  
    echo $value, "\n";  
}
```

```
1  
2  
3  
4  
5
```

Lea Estructuras de datos SPL en línea: <https://riptutorial.com/es/php/topic/6844/estructuras-de-datos-spl>

Capítulo 42: Examen de la unidad

Sintaxis

- [Lista completa de afirmaciones](#) . Ejemplos:

- `assertTrue(bool $condition[, string $messageIfFalse = '')];`
- `assertEquals(mixed $expected, mixed $actual[, string $messageIfNotEqual = '')];`

Observaciones

Unit pruebas Unit se utilizan para probar el código fuente para ver si contiene acuerdos con entradas como esperamos. Unit pruebas Unit son soportadas por la mayoría de los marcos. Hay varias pruebas diferentes de PHPUnit y pueden diferir en la sintaxis. En este ejemplo estamos usando PHPUnit .

Examples

Pruebas de reglas de clase

Digamos, tenemos un LoginForm simple de clase LoginForm con reglas () (utilizado en la página de inicio de sesión como plantilla de marco):

```
class LoginForm {
    public $email;
    public $rememberMe;
    public $password;

    /* rules() method returns an array with what each field has as a requirement.
     * Login form uses email and password to authenticate user.
     */
    public function rules() {
        return [
            // Email and Password are both required
            [['email', 'password'], 'required'],

            // Email must be in email format
            ['email', 'email'],

            // rememberMe must be a boolean value
            ['rememberMe', 'boolean'],

            // Password must match this pattern (must contain only letters and numbers)
            ['password', 'match', 'pattern' => '/^([a-zA-Z0-9]+)$/i'],
        ];
    }

    /**
     * validate function checks for correctness of the passed rules */
    public function validate($rule) {
        $success = true;
        list($var, $type) = $rule;
        foreach ((array) $var as $var) {
```

```

        switch ($type) {
            case "required":
                $success = $success && $this->$var != "";
                break;
            case "email":
                $success = $success && filter_var($this->$var, FILTER_VALIDATE_EMAIL);
                break;
            case "boolean":
                $success = $success && filter_var($this->$var, FILTER_VALIDATE_BOOLEAN,
FILTER_NULL_ON_FAILURE) !== null;
                break;
            case "match":
                $success = $success && preg_match($rule["pattern"], $this->$var);
                break;
            default:
                throw new \InvalidArgumentException("Invalid filter type passed")
        }
    }
    return $success;
}
}

```

Para realizar pruebas en esta clase, usamos pruebas de **unidad** (verificando el código fuente para ver si se ajusta a nuestras expectativas):

```

class LoginFormTest extends TestCase {
    protected $loginForm;

    // Executing code on the start of the test
    public function setUp() {
        $this->loginForm = new LoginForm;
    }

    // To validate our rules, we should use the validate() method

    /**
     * This method belongs to Unit test class LoginFormTest and
     * it's testing rules that are described above.
     */
    public function testRuleValidation() {
        $rules = $this->loginForm->rules();

        // Initialize to valid and test this
        $this->loginForm->email = "valid@email.com";
        $this->loginForm->password = "password";
        $this->loginForm->rememberMe = true;
        $this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");

        // Test email validation
        // Since we made email to be in email format, it cannot be empty
        $this->loginForm->email = '';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(empty)");

        // It does not contain "@" in string so it's invalid
        $this->loginForm->email = 'invalid.email.com';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(invalid format)");
    }
}

```

```

// Revert email to valid for next test
$this->loginForm->email = 'valid@email.com';

// Test password validation
// Password cannot be empty (since it's required)
$this->loginForm->password = '';
$this->assertFalse($this->loginForm->validate($rules), "Password should not be valid
(empty)");

// Revert password to valid for next test
$this->loginForm->password = 'ThisIsMyPassword';

// Test rememberMe validation
$this->loginForm->rememberMe = 999;
$this->assertFalse($this->loginForm->validate($rules), "RememberMe should not be valid
(integer type)");

// Revert remeberMe to valid for next test
$this->loginForm->rememberMe = true;
}

}

```

¿Cómo pueden ayudar exactamente las pruebas `Unit` (excluyendo ejemplos generales) aquí? Por ejemplo, encaja muy bien cuando obtenemos resultados inesperados. Por ejemplo, tomemos esta regla de antes:

```
[ 'password', 'match', 'pattern' => '/^[a-z0-9]+$/i' ],
```

En cambio, si nos perdemos una cosa importante y escribimos esto:

```
[ 'password', 'match', 'pattern' => '/^a-z0-9$/i' ],
```

Con docenas de reglas diferentes (asumiendo que estamos usando no solo correo electrónico y contraseña), es difícil detectar errores. Esta prueba unitaria:

```

// Initialize to valid and test this
$this->loginForm->email = "valid@email.com";
$this->loginForm->password = "password";
$this->loginForm->rememberMe = true;
$this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");

```

Pasará nuestro **primer** ejemplo pero no el **segundo**. ¿Por qué? Porque en el segundo ejemplo escribimos un patrón con un error tipográfico (signo + perdido), lo que significa que solo acepta una letra / número.

Las pruebas unitarias se pueden ejecutar en la consola con el comando: `phpunit [path_to_file]`. Si todo está bien, deberíamos poder ver que todas las pruebas están en estado `OK`, de lo contrario, veremos `Error` (errores de sintaxis) o `Fail` (al menos una línea en ese método no se aprobó).

Con parámetros adicionales como `--coverage` también podemos ver visualmente cuántas líneas en el código de back-end se probaron y cuáles pasaron / fallaron. Esto se aplica a cualquier marco

que haya instalado [PHPUnit](#).

Ejemplo de cómo se ve la prueba `PHPUnit` en la consola (apariencia general, no de acuerdo con este ejemplo):

```
vagrant@precise64:/var/www/phpunit-randomizer(master✓) * ./bin/phpunit-randomizer
PHPUnit 4.2.1 by Sebastian Bergmann.

Configuration read from /var/www/phpunit-randomizer/phpunit.xml.dist

. ExampleTest::test4
. ExampleTest::test3
. ExampleTest::test2
. ExampleTest::test5
. ExampleTest::test1
. OtherExampleTest::test4
. OtherExampleTest::test1
. OtherExampleTest::test3
. OtherExampleTest::test5
. OtherExampleTest::test2

Time: 151 ms, Memory: 3.50Mb
OK (10 tests, 0 assertions)

Randomized with seed: 8639
vagrant@precise64:/var/www/phpunit-randomizer(master✓) * ./bin/phpunit-randomizer
PHPUnit 4.2.1 by Sebastian Bergmann.

Configuration read from /var/www/phpunit-randomizer/phpunit.xml.dist
    testRunner.php
. ExampleTest::test2
. ExampleTest::test4
. ExampleTest::test1
. ExampleTest::test5
. ExampleTest::test3
. OtherExampleTest::test2
. OtherExampleTest::test1
. OtherExampleTest::test4
. OtherExampleTest::test3
. OtherExampleTest::test5

Time: 108 ms, Memory: 3.50Mb
OK (10 tests, 0 assertions)

Randomized with seed: 4674
```

PHPUnit Data Providers

Los métodos de prueba a menudo necesitan datos para ser probados. Para probar algunos métodos completamente, debe proporcionar diferentes conjuntos de datos para cada posible

condición de prueba. Por supuesto, puedes hacerlo manualmente usando bucles, como este:

```
...
public function testSomething()
{
    $data = [...];
    foreach($data as $dataSet) {
        $this->assertSomething($dataSet);
    }
}
...
```

Y alguien puede encontrarlo conveniente. Pero hay algunos inconvenientes de este enfoque. Primero, tendrá que realizar acciones adicionales para extraer datos si su función de prueba acepta varios parámetros. En segundo lugar, en caso de error, sería difícil distinguir el conjunto de datos con errores sin mensajes adicionales y depuración. En tercer lugar, PHPUnit proporciona una forma automática de manejar los conjuntos de datos de prueba utilizando [proveedores de datos](#).

El proveedor de datos es una función que debe devolver datos para su caso de prueba particular.

Un método de proveedor de datos debe ser público y devolver una **matriz de matrices** o un objeto que implementa la interfaz de **iterador** y **produce una matriz** para cada paso de iteración. Para cada matriz que forma parte de la colección, se llamará al método de prueba con el contenido de la matriz como sus argumentos.

Para usar un proveedor de datos con su prueba, use la anotación `@dataProvider` con el nombre de la función del proveedor de datos especificada:

```
/**
 * @dataProvider dataProviderForTest
 */
public function testEquals($a, $b)
{
    $this->assertEquals($a, $b);
}

public function dataProviderForTest()
{
    return [
        [1,1],
        [2,2],
        [3,2] //this will fail
    ];
}
```

Array de matrices

Tenga en cuenta que `dataProviderForTest()` devuelve una matriz de matrices. Cada matriz anidada tiene dos elementos y llenarán los parámetros necesarios para `testEquals()` uno por uno. Se lanzará un error como este. `Missing argument 2 for Test::testEquals()` si no hay suficientes elementos. PHPUnit pasará automáticamente

por los datos y ejecutará pruebas:

```
public function dataProviderForTest()
{
    return [
        [1,1], // [0] testEquals($a = 1, $b = 1)
        [2,2], // [1] testEquals($a = 2, $b = 2)
        [3,2]  // [2] There was 1 failure: 1) Test::testEquals with data set #2 (3, 4)
    ];
}
```

Cada conjunto de datos puede ser **nombrado** por conveniencia. Será más fácil detectar los datos que fallan:

```
public function dataProviderForTest()
{
    return [
        'Test 1' => [1,1], // [0] testEquals($a = 1, $b = 1)
        'Test 2' => [2,2], // [1] testEquals($a = 2, $b = 2)
        'Test 3' => [3,2] // [2] There was 1 failure:
                           //      1) Test::testEquals with data set "Test 3" (3, 4)
    ];
}
```

Iteradores

```
class MyIterator implements Iterator {
    protected $array = [];

    public function __construct($array) {
        $this->array = $array;
    }

    function rewind() {
        return reset($this->array);
    }

    function current() {
        return current($this->array);
    }

    function key() {
        return key($this->array);
    }

    function next() {
        return next($this->array);
    }

    function valid() {
        return key($this->array) !== null;
    }
}

class Test extends TestCase
{
```

```

/**
 * @dataProvider dataProviderForTest
 */
public function testEquals($a)
{
    $toCompare = 0;

    $this->assertEquals($a, $toCompare);
}

public function dataProviderForTest()
{
    return new MyIterator([
        'Test 1' => [0],
        'Test 2' => [false],
        'Test 3' => [null]
    ]);
}
}

```

Como puedes ver, el iterador simple también funciona.

Tenga en cuenta que incluso para un **solo** parámetro, el proveedor de datos debe devolver una matriz `[$parameter]`

Porque si cambiamos nuestro método `current()` (que en realidad devuelve datos en cada iteración) a esto:

```

function current() {
    return current($this->array)[0];
}

```

O cambiar los datos reales:

```

return new MyIterator([
    'Test 1' => 0,
    'Test 2' => false,
    'Test 3' => null
]);

```

Obtendremos un error:

```

There was 1 warning:

1) Warning
The data provider specified for Test::testEquals is invalid.

```

Por supuesto, no es útil usar el objeto `Iterator` sobre una matriz simple. Debería implementar alguna lógica específica para su caso.

Generadores

No se indica ni se muestra explícitamente en el manual, pero también puede usar un [generador](#)

como proveedor de datos. Tenga en cuenta que la clase `Generator` realmente implementa la interfaz `Iterator`.

Este es un ejemplo del uso de `DirectoryIterator` combinado con el `generator`:

```
/**  
 * @param string $file  
 *  
 * @dataProvider fileDataProvider  
 */  
public function testSomethingWithFiles($fileName)  
{  
    // $fileName is available here  
  
    // do test here  
}  
  
public function fileDataProvider()  
{  
    $directory = new DirectoryIterator('path-to-the-directory');  
  
    foreach ($directory as $file) {  
        if ($file->isFile() && $file->isReadable()) {  
            yield [$file->getPathname()]; // invoke generator here.  
        }  
    }  
}
```

Tenga en cuenta el `yield` proveedor es una matriz. En su lugar, recibirá una advertencia de proveedor de datos no válido.

Excepciones de prueba

Digamos que quieres probar el método que lanza una excepción

```
class Car  
{  
    /**  
     * @throws \Exception  
     */  
    public function drive()  
    {  
        throw new \Exception('Useful message', 1);  
    }  
}
```

Puede hacerlo encerrando la llamada al método en un bloque `try / catch` y haciendo afirmaciones sobre las propiedades del objeto de ejecución, pero más convenientemente puede usar métodos de afirmación de excepción. A partir de [PHPUnit 5.2](#), tiene métodos `expectException()` disponibles para confirmar el tipo de excepción, el mensaje y el código

```
class DriveTest extends PHPUnit_Framework_TestCase  
{  
    public function testDrive()  
    {
```

```

    // prepare
    $car = new \Car();
    $expectedClass = \Exception::class;
    $expectedMessage = 'Useful message';
    $expectedCode = 1;

    // test
    $this->expectException($expectedClass);
    $this->expectMessage($expectedMessage);
    $this->expectCode($expectedCode);

    // invoke
    $car->drive();
}
}

```

Si está utilizando una versión anterior de PHPUnit, el método `setExpectedException` puede usarse en lugar de los métodos `expectX()`, pero tenga en cuenta que está en desuso y se eliminará en la versión 6.

```

class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->setExpectedException($expectedClass, $expectedMessage, $expectedCode);

        // invoke
        $car->drive();
    }
}

```

Lea Examen de la unidad en línea: <https://riptutorial.com/es/php/topic/3417/examen-de-la-unidad>

Capítulo 43: Expresiones regulares (regexp / PCRE)

Sintaxis

- `preg_replace($pattern, $replacement, $subject, $limit = -1, $count = 0);`
- `preg_replace_callback($pattern, $callback, $subject, $limit = -1, $count = 0);`
- `preg_match($pattern, $subject, &$matches, $flags = 0, $offset = 0);`
- `preg_match_all($pattern, $subject, &$matches, $flags = PREG_PATTERN_ORDER, $offset = 0);`
- `preg_split($pattern, $subject, $limit = -1, $flags = 0)`

Parámetros

Parámetro	Detalles
<code>\$pattern</code>	una cadena con una expresión regular (patrón PCRE)

Observaciones

Las expresiones regulares de PHP siguen los estándares de patrones PCRE, que se derivan de las expresiones regulares de Perl.

Todas las cadenas PCRE en PHP deben incluirse entre delimitadores. Un delimitador puede ser cualquier carácter no alfanumérico, sin barra inversa, sin espacios en blanco. Los delimitadores populares son ~ , / , % por ejemplo.

Los patrones de PCRE pueden contener grupos, clases de caracteres, grupos de caracteres, aseveraciones anticipadas / anticipadas y personajes escapados.

Es posible usar modificadores PCRE en la cadena `$pattern`. Algunos de los más comunes son i (no distingue mayúsculas y minúsculas), m (multilínea) y s (el punto metacaracteriano incluye líneas nuevas). El modificador g (global) no está permitido, en su lugar usará la función `preg_match_all`.

Las coincidencias con las cadenas PCRE se realizan con \$ cadenas prefijadas numeradas:

```
<?php  
  
$replaced = preg_replace('%hello ([a-z]+) world%', 'goodbye $1 world', 'hello awesome world');  
  
echo $replaced; // 'goodbye awesome world'
```

Examples

Coincidencia de cadenas con expresiones regulares

`preg_match` comprueba si una cadena coincide con la expresión regular.

```
$string = 'This is a string which contains numbers: 12345';
$isMatched = preg_match('%^([a-zA-Z]+: [0-9]+)$%', $string);
var_dump($isMatched); // bool(true)
```

Si pasa un tercer parámetro, se rellenará con los datos coincidentes de la expresión regular:

```
preg_match('%^([a-zA-Z]+: ([0-9]+))$', 'This is a string which contains numbers: 12345',
$matches);
// $matches now contains results of the regular expression matches in an array.
echo json_encode($matches); // ["numbers": 12345, "numbers", "12345"]
```

`$matches` contiene una matriz de la concordancia completa y luego las subcadenas en la expresión regular delimitada por paréntesis, en el orden de desplazamiento de paréntesis abierto. Eso significa que, si tiene `/z(a(b))` como expresión regular, el índice 0 contiene la subcadena completa `zab`, el índice 1 contiene la subcadena delimitada por los paréntesis externos `ab` y el índice 2 contiene los paréntesis internos `b`.

Dividir cadena en matriz por una expresión regular

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";
// [0-9]: Any single character in the range 0 to 9
// +    : One or more of 0 to 9
$array = preg_split("/[0-9]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
//Or
// []   : Character class
// \d   : Any digit
// +   : One or more of Any digit
$array = preg_split("/[\d]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
```

Salida:

```
Array
(
    [0] => PHP
    [1] => CSS
    [2] => HTML
    [3] => AJAX
    [4] => JSON
)
```

Para dividir una cadena en una matriz, simplemente pase la cadena y una `preg_split()`; para `preg_split()`; para hacer coincidir y buscar, agregar un tercer parámetro (`limit`) le permite establecer el número de "coincidencias" que se realizarán, la cadena restante se agregará al final de la matriz.

El cuarto parámetro es (`flags`) aquí usamos el `PREG_SPLIT_NO_EMPTY` que evita que nuestra matriz

contenga cualquier clave / valor vacío.

Cadena sustituyendo con expresión regular.

```
$string = "a;b;c\nd;e;f";
// $1, $2 and $3 represent the first, second and third capturing groups
echo preg_replace("(^([^;]+);([^;]+);([^;]+)$)m", "$3;$2;$1", $string);
```

Salidas

```
c;b;a
f;e;d
```

Busca todo entre los puntos y comics e invierte el orden.

Partido RegExp global

Una coincidencia RegExp *global* se puede realizar utilizando `preg_match_all`. `preg_match_all` devuelve todos los resultados coincidentes en la cadena de asunto (a diferencia de `preg_match`, que solo devuelve el primero).

La función `preg_match_all` devuelve el número de coincidencias. Las `$matches` tercer parámetro `$matches` contendrán coincidencias en formato controlado por indicadores que se pueden dar en el cuarto parámetro.

Si se le da una matriz, `$matches` contendrá la matriz en un formato similar que obtendría con `preg_match`, excepto que `preg_match` detiene en la primera coincidencia, donde `preg_match_all` repite en la cadena hasta que la cadena se consume por completo y devuelve el resultado de cada iteración en una matriz multidimensional, cuyo formato puede ser controlado por la bandera en el cuarto argumento.

El cuarto argumento, `$flags`, controla la estructura de `$matches` array array. El modo predeterminado es `PREG_PATTERN_ORDER` y los posibles indicadores son `PREG_SET_ORDER` y `PREG_PATTERN_ORDER`.

El siguiente código demuestra el uso de `preg_match_all`:

```
$subject = "a1b c2d3e f4g";
$pattern = '/[a-z]([0-9])[a-z]/';

var_dump(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)); // int(3)
var_dump($matches);
preg_match_all($pattern, $subject, $matches); // the flag is PREG_PATTERN_ORDER by default
var_dump($matches);
// And for reference, same regexp run through preg_match()
preg_match($pattern, $subject, $matches);
var_dump($matches);
```

El primer `var_dump` de `PREG_SET_ORDER` da esta salida:

```

array(3) {
[0]=>
array(2) {
[0]=>
string(3) "alb"
[1]=>
string(1) "1"
}
[1]=>
array(2) {
[0]=>
string(3) "c2d"
[1]=>
string(1) "2"
}
[2]=>
array(2) {
[0]=>
string(3) "f4g"
[1]=>
string(1) "4"
}
}

```

`$matches` tiene tres matrices anidadas. Cada matriz representa una coincidencia, que tiene el mismo formato que el resultado de retorno de `preg_match`.

El segundo `var_dump` (`PREG_PATTERN_ORDER`) da esta salida:

```

array(2) {
[0]=>
array(3) {
[0]=>
string(3) "alb"
[1]=>
string(3) "c2d"
[2]=>
string(3) "f4g"
}
[1]=>
array(3) {
[0]=>
string(1) "1"
[1]=>
string(1) "2"
[2]=>
string(1) "4"
}
}

```

Cuando la misma expresión regular se ejecuta a través de `preg_match`, se devuelve la siguiente matriz:

```

array(2) {
[0] =>
string(3) "alb"
[1] =>
string(1) "1"
}

```

```
}
```

Cadena reemplazar con devolución de llamada

`preg_replace_callback` funciona enviando cada grupo de captura coincidente a la devolución de llamada definida y la reemplaza con el valor de retorno de la devolución de llamada. Esto nos permite reemplazar cadenas basadas en cualquier tipo de lógica.

```
$subject = "He said 123abc, I said 456efg, then she said 789hij";
$regex = "/\b(\d+)\w+/"';

// This function replaces the matched entries conditionally
// depending upon the first character of the capturing group
function regex_replace($matches) {
    switch($matches[1][0]) {
        case '7':
            $replacement = "<b>{$matches[0]}</b>";
            break;
        default:
            $replacement = "<i>{$matches[0]}</i>";
    }
    return $replacement;
}

$replaced_str = preg_replace_callback($regex, "regex_replace", $subject);

print_r($replaced_str);
# He said <i>123abc</i>, I said <i>456efg</i>, then she said <b>789hij</b>
```

Lea Expresiones regulares (regexp / PCRE) en línea:

<https://riptutorial.com/es/php/topic/852/expresiones-regulares--regexp---pcre->

Capítulo 44: Extensión de roscado múltiple

Observaciones

Con `pthread v3` solo se puede cargar cuando se usa el `cli` SAPI, por lo que es una buena práctica mantener la directiva `extension=php_pthreads.dll` en `php.ini` SOLAMENTE, si está utilizando PHP7 y Pthreads v3.

Si está utilizando **Wamp** en **Windows**, debe configurar la extensión en `php.ini`:

Abre `php\php.ini` y agrega:

```
extension=php_pthreads.dll
```

Con respecto a **los usuarios de Linux**, debe reemplazar `.dll` por `.so`:

```
extension=php_pthreads.so
```

Puede ejecutar este comando directamente para agregarlo a `php.ini` (cambie `/etc/php.ini` con su ruta personalizada)

```
echo "extension=php_pthreads.so" >> /etc/php.ini
```

Examples

Empezando

Para comenzar con subprocessos múltiples, necesitaría el `pthread-ext` para php, que puede ser instalado por

```
$ pecl install pthreads
```

y añadiendo la entrada a `php.ini`.

Un ejemplo simple:

```
<?php
// NOTE: Code uses PHP7 semantics.
class MyThread extends Thread {
    /**
     * @var string
     * Variable to contain the message to be displayed.
     */
    private $message;

    public function __construct(string $message) {
        // Set the message value for this particular instance.
```

```

        $this->message = $message;
    }

    // The operations performed in this function is executed in the other thread.
    public function run() {
        echo $this->message;
    }
}

// Instantiate MyThread
$myThread = new MyThread("Hello from an another thread!");
// Start the thread. Also it is always a good practice to join the thread explicitly.
// Thread::start() is used to initiate the thread,
$myThread->start();
// and Thread::join() causes the context to wait for the thread to finish executing
$myThread->join();

```

Uso de piscinas y trabajadores

La agrupación proporciona una abstracción de mayor nivel de la funcionalidad de Worker, incluida la gestión de referencias en la forma requerida por pthreads. De: <http://php.net/manual/en/class.pool.php>

Los grupos y los trabajadores proporcionan un mayor nivel de control y facilidad de creación de subprocessos múltiples

```

<?php
// This is the *Work* which would be ran by the worker.
// The work which you'd want to do in your worker.
// This class needs to extend the \Threading or \Collectable or \Thread class.
class AwesomeWork extends Thread {
    private $workName;

    /**
     * @param string $workName
     * The work name which would be given to every work.
     */
    public function __construct(string $workName) {
        // The block of code in the constructor of your work,
        // would be executed when a work is submitted to your pool.

        $this->workName = $workName;
        printf("A new work was submitted with the name: %s\n", $workName);
    }

    public function run() {
        // This block of code in, the method, run
        // would be called by your worker.
        // All the code in this method will be executed in another thread.
        $workName = $this->workName;
        printf("Work named %s starting...\n", $workName);
        printf("New random number: %d\n", mt_rand());
    }
}

// Create an empty worker for the sake of simplicity.
class AwesomeWorker extends Worker {
    public function run() {

```

```

    // You can put some code in here, which would be executed
    // before the Work's are started (the block of code in the `run` method of your Work)
    // by the Worker.
    /* ... */
}

}

// Create a new Pool Instance.
// The ctor of \Pool accepts two parameters.
// First: The maximum number of workers your pool can create.
// Second: The name of worker class.
$pool = new \Pool(1, \AwesomeWorker::class);

// You need to submit your jobs, rather the instance of
// the objects (works) which extends the \Threaded class.
$pool->submit(new \AwesomeWork("DeadlyWork"));
$pool->submit(new \AwesomeWork("FatalWork"));

// We need to explicitly shutdown the pool, otherwise,
// unexpected things may happen.
// See: http://stackoverflow.com/a/23600861/23602185
$pool->shutdown();

```

Lea Extensión de roscado múltiple en línea: <https://riptutorial.com/es/php/topic/1583/extension-de-roscado-multiple>

Capítulo 45: Filtros y funciones de filtro

Introducción

Esta extensión filtra los datos mediante la validación o la desinfección. Esto es especialmente útil cuando la fuente de datos contiene datos desconocidos (o extraños), como la entrada proporcionada por el usuario. Por ejemplo, estos datos pueden provenir de un formulario HTML.

Sintaxis

- `filter_var` mixto (variable \$ variable [, int \$ filtro = FILTER_DEFAULT [, \$ opciones mezcladas]])

Parámetros

Parámetro	Detalles
variable	Valor para filtrar. Tenga en cuenta que los valores escalares se convierten en cadena internamente antes de filtrarlos.
filtrar	El ID del filtro a aplicar. La página de manual Tipos de filtros enumera los filtros disponibles. Si se omite, se utilizará FILTER_DEFAULT, que es equivalente a FILTER_UNSAFE_RAW. Esto dará lugar a que no se realice ningún filtrado de forma predeterminada.
opciones	Matriz asociativa de opciones o disyunción bit a bit de banderas. Si el filtro acepta opciones, se pueden proporcionar indicadores en el campo "indicadores" de la matriz. Para el filtro de "devolución de llamada", se debe pasar el tipo llamable. La devolución de llamada debe aceptar un argumento, el valor que se filtrará y devolver el valor después de filtrarlo / sanearlo.

Examples

Validar correo electrónico

Al filtrar una dirección de correo electrónico, `filter_var()` devolverá los datos filtrados, en este caso la dirección de correo electrónico, o falso si no se puede encontrar una dirección de correo electrónico válida:

```
var_dump(filter_var('john@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('notValidEmail', FILTER_VALIDATE_EMAIL));
```

Resultados:

```
string(16) "john@example.com"
bool(false)
```

Esta función no valida los caracteres no latinos. El nombre de dominio internacionalizado se puede validar en su forma `xn--`.

Tenga en cuenta que no puede saber si la dirección de correo electrónico es correcta antes de enviarle un correo electrónico. Es posible que desee realizar algunas verificaciones adicionales, como verificar un registro MX, pero esto no es necesario. Si envía un correo electrónico de confirmación, no olvide eliminar las cuentas no utilizadas después de un breve período.

Validar un valor es un entero

Al filtrar un valor que debería ser un entero `filter_var()` devolverá los datos filtrados, en este caso el entero, o falso si el valor no es un entero. Los flotadores *no* son enteros:

```
var_dump(filter_var('10', FILTER_VALIDATE_INT));
var_dump(filter_var('a10', FILTER_VALIDATE_INT));
var_dump(filter_var('10a', FILTER_VALIDATE_INT));
var_dump(filter_var(' ', FILTER_VALIDATE_INT));
var_dump(filter_var('10.00', FILTER_VALIDATE_INT));
var_dump(filter_var('10,000', FILTER_VALIDATE_INT));
var_dump(filter_var('-5', FILTER_VALIDATE_INT));
var_dump(filter_var('+7', FILTER_VALIDATE_INT));
```

Resultados:

```
int(10)
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
int(-5)
int(7)
```

Si está esperando solo dígitos, puede usar una expresión regular:

```
if(is_string($_GET['entry']) && preg_match('#^0-9]+$#', $_GET['entry']))
    // this is a digit (positive) integer
else
    // entry is incorrect
```

Si convierte este valor en un entero, no tiene que hacer esta comprobación y, por lo tanto, puede usar `filter_var`.

Validando un número entero cae en un rango

Al validar que un entero cae dentro de un rango, la verificación incluye los límites mínimo y máximo:

```
$options = array(
    'options' => array(
        'min_range' => 5,
        'max_range' => 10,
    )
);
var_dump(filter_var('5', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('10', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('8', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('4', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('11', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('-6', FILTER_VALIDATE_INT, $options));
```

Resultados:

```
int(5)
int(10)
int(8)
bool(false)
bool(false)
bool(false)
```

Validar una URL

Al filtrar una URL, `filter_var()` devolverá los datos filtrados, en este caso la URL, o `false` si no se puede encontrar una URL válida:

URL: example.com

```
var_dump(filter_var('example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
```

URL: http://example.com

```
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
```

```
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));  
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(18) "http://example.com"  
string(18) "http://example.com"  
string(18) "http://example.com"  
bool(false)  
bool(false)
```

URL: <http://www.example.com>

```
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL));  
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,  
FILTER_FLAG_SCHEME_REQUIRED));  
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,  
FILTER_FLAG_HOST_REQUIRED));  
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,  
FILTER_FLAG_PATH_REQUIRED));  
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,  
FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(22) "http://www.example.com"  
string(22) "http://www.example.com"  
string(22) "http://www.example.com"  
bool(false)  
bool(false)
```

URL: <http://www.example.com/path/to/dir/>

```
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL));  
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,  
FILTER_FLAG_SCHEME_REQUIRED));  
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,  
FILTER_FLAG_HOST_REQUIRED));  
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,  
FILTER_FLAG_PATH_REQUIRED));  
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,  
FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(35) "http://www.example.com/path/to/dir/"  
string(35) "http://www.example.com/path/to/dir/"  
string(35) "http://www.example.com/path/to/dir/"  
string(35) "http://www.example.com/path/to/dir/"  
bool(false)
```

URL: <http://www.example.com/path/to/dir/index.php>

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
bool(false)
```

URL: <http://www.example.com/path/to/dir/index.php?test=y>

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
```

Advertencia : debe verificar el protocolo para protegerse contra un ataque XSS:

```
var_dump(filter_var('javascript://comment%0Aalert(1)', FILTER_VALIDATE_URL));
// string(31) "javascript://comment%0Aalert(1)"
```

Desinfectar filtros

Podemos usar filtros para desinfectar nuestra variable de acuerdo a nuestra necesidad.

Ejemplo

```
$string = "<p>Example</p>";
$newstring = filter_var($string, FILTER_SANITIZE_STRING);
var_dump($newstring); // string(7) "Example"
```

Lo anterior eliminará las etiquetas html de la variable `$string`.

Validación de valores booleanos

```
var_dump(filter_var(true, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(false, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(1, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(0, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('1', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('0', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(' ', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('true', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('false', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var([], FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // NULL
var_dump(filter_var(null, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
```

Validar un número es un flotador

Valida el valor como flotante y se convierte en flotante en caso de éxito.

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT));

var_dump(filter_var(1, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Resultados

```
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
```

```
float(1)
float(1.00001)
bool(false)
bool(false)
bool(false)
bool(false)

float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1)
float(1.00001)
float(1000)
float(1000)
float(1000)
float(1000.00001)
```

Validar una dirección MAC

Valida un valor es una dirección MAC válida

```
var_dump(filter_var('FA-F9-DD-B2-5E-0D', FILTER_VALIDATE_MAC));
var_dump(filter_var('DC-BB-17-9A-CE-81', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40-AB', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40', FILTER_VALIDATE_MAC));
var_dump(filter_var('', FILTER_VALIDATE_MAC));
```

Resultados:

```
string(17) "FA-F9-DD-B2-5E-0D"
string(17) "DC-BB-17-9A-CE-81"
string(17) "96-D5-9E-67-40-AB"
bool(false)
bool(false)
```

Sanitze Direcciones de correo electrónico

Elimine todos los caracteres excepto las letras, los dígitos y! # \$% & * + - = ? ^ _ ` {} ~ @. [].

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~.[]@example.com", FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john/@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john\@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('joh n@example.com', FILTER_SANITIZE_EMAIL));
```

Resultados:

```
string(16) "john@example.com"
string(33) "!#$%&'*+-=?^_`{|}~.[]@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
```

```
string(16) "john@example.com"
```

Desinfectar enteros

Eliminar todos los caracteres, excepto los dígitos, el signo más y el signo menos.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var("!#$%&'*+-=?^`{|}~@.[]0123456789abcdefghijklmnopqrstuvwxyz",
FILTER_SANITIZE_NUMBER_INT));
```

Resultados:

```
string(1) "1"
string(2) "-1"
string(1) "1"
string(1) "1"
string(1) "1"
string(2) "-1"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(3) "100"
string(4) "+100"
string(4) "-100"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(12) "+-0123456789"
```

Desinfectar URL

URLs de Sanitze

Elimine todos los caracteres excepto letras, dígitos y \$ -_. +! * '(), {} | \ ^ ~ [] ` <> #%" ; /?: @ & =

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_SANITIZE_URL));
var_dump(filter_var("http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-
=?^`{|}~@.[]", FILTER_SANITIZE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=a b c',
FILTER_SANITIZE_URL));
```

Resultados:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(72) "http://www.example.com/path/to/dir/index.php?test=y!#$%&*'+=?^_`{|}~.[]"
string(53) "http://www.example.com/path/to/dir/index.php?test=abc"
```

Desinfectar flotadores

Elimine todos los caracteres excepto los dígitos, + - y opcionalmente, eE.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT));
```

Resultados:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(9) "18281-009"
```

Con la opción `FILTER_FLAG_ALLOW_THOUSAND`:

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Resultados:

```
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(5) "1,000"
string(6) "1,0000"
string(9) "1,0000000"
string(10) "1,00000001"
string(9) "18281-009"
```

Con la opción FILTER_FLAG_ALLOW_SCIENTIFIC :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
```

Resultados:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(10) "18281e-009"
```

Validar direcciones IP

Valida un valor es una dirección IP válida

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0001', FILTER_VALIDATE_IP));
```

```
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP));
```

Resultados:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Valide una dirección IP válida de IPv4:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV4));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
```

Resultados:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Valide una dirección IP válida de IPv6:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV6));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

Resultados:

```
bool(false)
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
bool(false)
```

Validar una dirección IP no está en un rango privado:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
```

Resultados:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
```

```
bool(false)
string(9) "127.0.0.1"
```

Validar una dirección IP no está en un rango reservado:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
```

Resultados:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
bool(false)
```

Lea Filtros y funciones de filtro en línea: <https://riptutorial.com/es/php/topic/1679/filtros-y-funciones-de-filtro>

Capítulo 46: Formato de cadena

Examples

Extracción / sustitución de subcadenas.

Los caracteres individuales se pueden extraer mediante la sintaxis de matriz (corchete cuadrado) y la sintaxis de corchete. Estas dos sintaxis solo devolverán un solo carácter de la cadena. Si se necesita más de un carácter, se requerirá una función, es decir, `substr`

Las cadenas, como todo en PHP, están `0`-indexadas.

```
$foo = 'Hello world';

$foo[6]; // returns 'w'
$foo{6}; // also returns 'w'

substr($foo, 6, 1); // also returns 'w'
substr($foo, 6, 2); // returns 'wo'
```

Las cadenas también se pueden cambiar de un carácter a la vez con la misma sintaxis de corchete y corsé. Reemplazar más de un carácter requiere una función, es decir, `substr_replace`

```
$foo = 'Hello world';

$foo[6] = 'W'; // results in $foo = 'Hello World'
$foo{6} = 'W'; // also results in $foo = 'Hello World'

substr_replace($foo, 'Whi', 6, 2); // results in 'Hello Whirled'
// note that the replacement string need not be the same length as the substring replaced
```

Interpolación de cuerdas

También puede utilizar la interpolación para insertar (*insertar*) una variable dentro de una cadena. La interpolación funciona solo en cadenas entre comillas y la sintaxis heredoc.

```
$name = 'Joel';

// $name will be replaced with `Joel`
echo "<p>Hello $name, Nice to see you.</p>";
#           ^
#>   "<p>Hello Joel, Nice to see you.</p>"

// Single Quotes: outputs $name as the raw text (without interpreting it)
echo 'Hello $name, Nice to see you.'; # Careful with this notation
#> "Hello $name, Nice to see you."
```

El formato de **sintaxis compleja (rizado)** proporciona otra opción que requiere que ajuste su variable entre llaves `{}`. Esto puede ser útil cuando incrusta variables dentro del contenido textual

y ayuda a prevenir una posible ambigüedad entre el contenido textual y las variables.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo "<p>We need more {$name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>

// This line will throw an error (as '$names' is not defined)
echo "<p>We need more $names to help us!</p>";
#> "Notice: Undefined variable: names"
```

La sintaxis {} solo interpola las variables que comienzan con \$ en una cadena. La sintaxis {} no evalúa expresiones PHP arbitrarias.

```
// Example trying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
#> "1 + 2 = {1 + 2}"

// Example using a constant
define("HELLO_WORLD", "Hello World!!");
echo "My constant is {$HELLO_WORLD}";
#> "My constant is {HELLO_WORLD}"

// Example using a function
function say_hello() {
    return "Hello!";
}
echo "I say: {say_hello()}";
#> "I say: {say_hello()}"
```

Sin embargo, la sintaxis {} evalúa el acceso a la matriz, el acceso a la propiedad y las llamadas a funciones / métodos en variables, elementos de la matriz o propiedades:

```
// Example accessing a value from an array – multidimensional access is allowed
$companions = [0 => ['name' => 'Amy Pond'], 1 => ['name' => 'Dave Random']];
echo "The best companion is: {$companions[0]['name']}";
#> "The best companion is: Amy Pond"

// Example of calling a method on an instantiated object
class Person {
    function say_hello() {
        return "Hello!";
    }
}

$max = new Person();

echo "Max says: {$max->say_hello()}";
#> "Max says: Hello!"

// Example of invoking a Closure – the parameter list allows for custom expressions
$greet = function($num) {
    return "A $num greetings!";
};
echo "From us all: {$greet(10 ** 3)}";
#> "From us all: A 1000 greetings!"
```

Observe que el signo de \$ dólar puede aparecer después de la llave de apertura { como en los ejemplos anteriores, o, como en Perl o Shell Script, puede aparecer delante de él:

```
$name = 'Joel';

// Example using the curly brace syntax with dollar sign before the opening curly brace
echo "<p>We need more ${name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"
```

La Complex (curly) syntax no se llama como tal porque es compleja, sino más bien porque permite el uso de '**expresiones complejas**'. [Leer más sobre Complex \(curly\) syntax](#)

Lea Formato de cadena en línea: <https://riptutorial.com/es/php/topic/6696/formato-de-cadena>

Capítulo 47: Funciones

Sintaxis

- function func_name (\$ parametersName1, \$ parametersName2) {code_to_run (); }
- function func_name (\$ optionalParameter = default_value) {code_to_run (); }
- function func_name (type_name \$ parametersName) {code_to_run (); }
- function & returns_by_reference () {code_to_run (); }
- function func_name (& \$ referenceParameter) {code_to_run (); }
- function func_name (... \$ variadicParameters) {code_to_run (); } // PHP 5.6+
- function func_name (type_name & ... \$ varRefParams) {code_to_run (); } // PHP 5.6+
- function func_name (): return_type {code_to_run (); } // PHP 7.0+

Examples

Uso básico de la función

Una función básica se define y ejecuta así:

```
function hello($name)
{
    print "Hello $name";
}

hello("Alice");
```

Parámetros opcionales

Las funciones pueden tener parámetrosopcionales, por ejemplo:

```
function hello($name, $style = 'Formal')
{
    switch ($style) {
        case 'Formal':
            print "Good Day $name";
            break;
        case 'Informal':
            print "Hi $name";
            break;
        case 'Australian':
            print "G'day $name";
            break;
        default:
            print "Hello $name";
            break;
    }
}

hello('Alice');
// Good Day Alice
```

```
hello('Alice', 'Australian');
// G'day Alice
```

Pasando Argumentos por Referencia

Los argumentos de la función se pueden pasar "Por referencia", lo que permite a la función modificar la variable utilizada fuera de la función:

```
function pluralize(&$word)
{
    if (substr($word, -1) == 'y') {
        $word = substr($word, 0, -1) . 'ies';
    } else {
        $word .= 's';
    }
}

$word = 'Bannana';
pluralize($word);

print $word;
// Bannanas
```

Los argumentos de objeto siempre se pasan por referencia:

```
function addOneDay($date)
{
    $date->modify('+1 day');
}

$date = new DateTime('2014-02-28');
addOneDay($date);

print $date->format('Y-m-d');
// 2014-03-01
```

Para evitar la transferencia implícita de un objeto por referencia, debe `clone` el objeto.

Pasar por referencia también se puede utilizar como una forma alternativa de devolver parámetros. Por ejemplo, la función `socket_getpeername`:

```
bool socket_getpeername ( resource $socket , string &$address [, int &$port ] )
```

Este método en realidad apunta a devolver la dirección y el puerto del par, pero como hay dos valores para devolver, elige usar parámetros de referencia en su lugar. Se puede llamar así:

```
if(!socket_getpeername($socket, $address, $port)) {
    throw new RuntimeException(socket_last_error());
}
echo "Peer: $address:$port\n";
```

Las variables `$address` y `$port` no necesitan definirse antes. Lo harán:

1. ser definido como `null` primero,
2. luego pasa a la función con el valor `null` predefinido
3. luego modificado en la función
4. terminan definidos como la dirección y el puerto en el contexto de llamada.

Listas de argumentos de longitud variable

5.6

PHP 5.6 introdujo listas de argumentos de longitud variable (también conocidos como varargs, argumentos variadic), usando el token `...` antes del nombre del argumento para indicar que el parámetro es variadic, es decir, es una matriz que incluye todos los parámetros suministrados desde ese en adelante.

```
function variadic_func($nonVariadic, ...$variadic) {
    echo json_encode($variadic);
}

variadic_func(1, 2, 3, 4); // prints [2,3,4]
```

Los nombres de los tipos se pueden agregar delante de `...`

```
function foo(Bar ...$bars) {}
```

El operador `&` reference se puede agregar antes de `...`, pero después del nombre de tipo (si corresponde). Considera este ejemplo:

```
class Foo{}
function a(Foo &...$foos) {
    $i = 0;
    foreach($a as &$foo){ // note the &
        $foo = $i++;
    }
}
$a = new Foo;
$c = new Foo;
$b =& $c;
a($a, $b);
var_dump($a, $b, $c);
```

Salida:

```
int(0)
int(1)
int(1)
```

Por otro lado, una matriz (`o Traversable`) de argumentos se puede desempaquetar para pasar a una función en forma de una lista de argumentos:

```
var_dump(...hash_algos());
```

Salida:

```
string(3) "md2"
string(3) "md4"
string(3) "md5"
...
```

Compare con este fragmento sin usar ... :

```
var_dump(hash_algos());
```

Salida:

```
array(46) {
    [0]=>
    string(3) "md2"
    [1]=>
    string(3) "md4"
    ...
}
```

Por lo tanto, las funciones de redireccionamiento para funciones variad ahora se pueden hacer fácilmente, por ejemplo:

```
public function formatQuery($query, ...$args){
    return sprintf($query, ...array_map([$mysqli, "real_escape_string"], $args));
}
```

Además de las matrices, también se pueden usar los `Traversable`, como `Iterator` (especialmente muchas de sus subclases de SPL). Por ejemplo:

```
$iterator = new LimitIterator(new ArrayIterator([0, 1, 2, 3, 4, 5, 6]), 2, 3);
echo bin2hex(pack("c*", ...$it)); // Output: 020304
```

Si el iterador itera infinitamente, por ejemplo:

```
$iterator = new InfiniteIterator(new ArrayIterator([0, 1, 2, 3, 4]));
var_dump(...$iterator);
```

Diferentes versiones de PHP se comportan de manera diferente:

- Desde PHP 7.0.0 hasta PHP 7.1.0 (beta 1):
 - Se producirá una falla de segmentación
 - El proceso de PHP saldrá con el código 139.
- En PHP 5.6:
 - Se mostrará un error fatal de agotamiento de la memoria ("Tamaño de memoria permitido de % d bytes agotados").
 - El proceso de PHP saldrá con el código 255

Nota: HHVM (v3.10 - v3.12) no admite el desempaquetado de `Traversable`s. En este

intentó se mostrará un mensaje de advertencia "Sólo se pueden desempaquetar los contenedores".

Alcance de la función

Las variables dentro de las funciones están dentro de un ámbito local como este.

```
$number = 5
function foo() {
    $number = 10
    return $number
}

foo(); //Will print 10 because text defined inside function is a local variable
```

Lea Funciones en línea: <https://riptutorial.com/es/php/topic/4551/funciones>

Capítulo 48: Funciones de hash de contraseña

Introducción

A medida que los servicios web más seguros evitan el almacenamiento de contraseñas en formato de texto plano, los lenguajes como PHP proporcionan varias funciones hash (no descifrables) para admitir el estándar de la industria más seguro. Este tema proporciona documentación para el hashing adecuado con PHP.

Sintaxis

- `string password_hash (string $password , integer $algo [, array $options])`
- `boolean password_verify (string $password , string $hash)`
- `boolean password_needs_rehash (string $hash , integer $algo [, array $options])`
- `array password_get_info (string $hash)`

Observaciones

Antes de PHP 5.5, puede usar [el paquete de compatibilidad](#) para proporcionar las funciones `password_*`. Se recomienda encarecidamente que utilice el paquete de compatibilidad si puede hacerlo.

Con o sin el paquete de compatibilidad, [la funcionalidad correcta de Bcrypt a través de `crypt\(\)` basa en PHP 5.3.7+](#). De lo contrario, *debe* restringir las contraseñas a conjuntos de caracteres solo ASCII.

Nota: si usa PHP 5.5 o una [versión inferior](#), está usando una [versión no compatible de PHP](#) que ya no recibe actualizaciones de seguridad. Actualice tan pronto como sea posible, puede actualizar sus hashes de contraseña posteriormente.

Selección de algoritmo

Algoritmos seguros

- **bcrypt** es su mejor opción siempre que use el estiramiento de teclas para aumentar el tiempo de cálculo del hash, ya que hace que los [ataques de fuerza bruta sean extremadamente lentos](#).
- **argon2** es otra opción que [estará disponible en PHP 7.2](#).

Algoritmos inseguros

Los siguientes algoritmos de hashing son **inseguros o no aptos para el propósito** y, por lo tanto, **no deben utilizarse**. Nunca fueron adecuados para el hashing de contraseñas, ya que están

diseñados para resúmenes rápidos en lugar de hashes de contraseñas lentas y difíciles de aplicar.

Si utiliza alguno de ellos, incluso las sales, debe **cambiar** a uno de los algoritmos de seguridad recomendados **lo antes posible**.

Algoritmos considerados inseguros:

- **MD4** - ataque de colisión encontrado en 1995
- **MD5** - ataque de colisión encontrado en 2005
- **SHA-1** - ataque de colisión demostrado en 2015

Algunos algoritmos pueden usarse de manera segura como algoritmo de resumen de mensajes para probar la autenticidad, pero **nunca como algoritmo de hashing de contraseña**:

- **SHA-2**
- **SHA-3**

Tenga en cuenta que los hashes fuertes como SHA256 y SHA512 son ininterrumpidos y robustos, sin embargo, en general es más seguro usar las funciones hash **bcrypt** o **argon2**, ya que los ataques de fuerza bruta contra estos algoritmos son mucho más difíciles para las computadoras clásicas.

Examples

Determine si un hash de contraseña existente puede actualizarse a un algoritmo más fuerte

Si está utilizando el método `PASSWORD_DEFAULT` para permitir que el sistema elija el mejor algoritmo para cifrar sus contraseñas, ya que la fuerza predeterminada aumenta, es posible que desee volver a borrar las contraseñas antiguas a medida que los usuarios inician sesión

```
<?php
// first determine if a supplied password is valid
if (password_verify($plaintextPassword, $hashedPassword)) {

    // now determine if the existing hash was created with an algorithm that is
    // no longer the default
    if (password_needs_rehash($hashedPassword, PASSWORD_DEFAULT)) {

        // create a new hash with the new default
        $newHashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT);

        // and then save it to your data store
        //$db->update(...);

    }
}
?>
```

Si las funciones `password_*` no están disponibles en su sistema (y no puede usar el paquete de compatibilidad vinculado en las observaciones a continuación), puede determinar el algoritmo y

usarlo para crear el hash original en un método similar al siguiente:

```
<?php
if (substr($hashedPassword, 0, 4) == '$2y$' && strlen($hashedPassword) == 60) {
    echo 'Algorithm is Bcrypt';
    // the "cost" determines how strong this version of Bcrypt is
    preg_match('/^$2y\$(\d+)\$/i', $hashedPassword, $matches);
    $cost = $matches[1];
    echo 'Bcrypt cost is '.$cost;
}
?>
```

Creando un hash de contraseña

Cree hashes de contraseñas utilizando [password_hash\(\)](#) para usar el hash estándar o la derivación de claves más recomendables de la industria. Al momento de escribir, el estándar es [bcrypt](#), lo que significa que `PASSWORD_DEFAULT` contiene el mismo valor que `PASSWORD_BCRYPT`.

```
$options = [
    'cost' => 12,
];

$hashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT, $options);
```

El tercer parámetro **no es obligatorio**.

El valor del `'cost'` debe elegirse en función del hardware de su servidor de producción. Incrementarlo hará que la contraseña sea más costosa de generar. Cuanto más costoso es generar, más tiempo tomará cualquiera que intente descifrarlo para generarlo también. Lo ideal es que el costo sea lo más alto posible, pero en la práctica debe establecerse para que no disminuya la velocidad demasiado. En algún lugar entre 0.1 y 0.4 segundos estaría bien. Utilice el valor predeterminado si tiene dudas.

5.5

En PHP inferior a 5.5.0, las funciones `password_*` no están disponibles. Debe usar [el paquete de compatibilidad](#) para sustituir esas funciones. Tenga en cuenta que el paquete de compatibilidad requiere PHP 5.3.7 o superior o una versión que [solución de fixport](#) (como la que proporciona RedHat).

Si no puede usarlos, puede implementar el hashing de contraseñas con [crypt\(\)](#). Como `password_hash()` se implementa como un envoltorio alrededor de la función `crypt()`, no necesita perder ninguna funcionalidad.

```
// this is a simple implementation of a bcrypt hash otherwise compatible
// with `password_hash()`
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation

// if `CRYPT_BLOWFISH` is 1, that means bcrypt (which uses blowfish) is available
// on your system
if (CRYPT_BLOWFISH == 1) {
```

```

$salt = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
$salt = base64_encode($salt);
// crypt uses a modified base64 variant
$source = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
$dest = './ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
$salt = strstr(rtrim($salt, '='), $source, $dest);
$salt = substr($salt, 0, 22);
// `crypt()` determines which hashing algorithm to use by the form of the salt string
// that is passed in
$hashedPassword = crypt($plaintextPassword, '$2y$10$' . $salt . '$');
}

```

Sal para el hash de contraseña

A pesar de la fiabilidad del algoritmo de cripta, todavía existe una vulnerabilidad frente a las [tablas de arco iris](#). Esa es la razón, por eso se recomienda usar **sal**.

Un salt es algo que se agrega a la contraseña antes de hacer hashing para hacer que la cadena fuente sea única. Dadas dos contraseñas idénticas, los hashes resultantes también serán únicos, porque sus sales son únicas.

Una sal aleatoria es una de las piezas más importantes de la seguridad de su contraseña. Esto significa que incluso con una tabla de búsqueda de hashes de contraseña conocida, un atacante no puede hacer coincidir el hash de contraseña del usuario con los hashes de contraseña de la base de datos, ya que se ha utilizado un salt aleatorio. Debes usar sales siempre aleatorias y criptográficamente seguras. [Lee mas](#)

Con `password_hash()` `bcrypt` algorithm, la sal de texto sin formato se almacena junto con el hash resultante, lo que significa que el hash se puede transferir a través de diferentes sistemas y plataformas y aún puede compararse con la contraseña original.

7.0

Incluso cuando esto no se recomienda, puede usar la opción de `salt` para definir su propia sal aleatoria.

```

$options = [
    'salt' => $salt, //see example below
];

```

Importante Si omite esta opción, `password_hash()` generará un valor aleatorio de sal para cada hash de contraseña. Este es el modo de operación previsto.

7.0

La opción `salt` ha sido [desaprobada](#) a partir de PHP 7.0.0. Ahora se prefiere usar simplemente la sal que se genera de forma predeterminada.

Verificando una contraseña contra un hash

```
password_verify()
```

es la función incorporada provista (a partir de PHP 5.5) para verificar la validez de una contraseña contra un hash conocido.

```
<?php
if (password_verify($plaintextPassword, $hashedPassword)) {
    echo 'Valid Password';
}
else {
    echo 'Invalid Password.';
}
?>
```

Todos los algoritmos de hash admitidos almacenan información que identifica qué hash se usó en el hash mismo, por lo que no es necesario indicar con qué algoritmo está utilizando para codificar la contraseña de texto simple.

Si las funciones `password_*` no están disponibles en su sistema (y no puede usar el paquete de compatibilidad vinculado en las observaciones a continuación), puede implementar la verificación de contraseña con la función `crypt()`. Tenga en cuenta que deben tomarse precauciones específicas para evitar [los ataques de tiempo](#).

```
<?php
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation
if (CRYPT_BLOWFISH == 1) {
    // `crypt()` discards all characters beyond the salt length, so we can pass in
    // the full hashed password
    $hashedCheck = crypt($plaintextPassword, $hashedPassword);

    // this is a basic constant-time comparison based on the full implementation used
    // in `password_hash()`
    $status = 0;
    for ($i=0; $i<strlen($hashedCheck); $i++) {
        $status |= (ord($hashedCheck[$i]) ^ ord($hashedPassword[$i]));
    }

    if ($status === 0) {
        echo 'Valid Password';
    }
    else {
        echo 'Invalid Password';
    }
}
?>
```

Lea Funciones de hash de contraseña en línea: <https://riptutorial.com/es/php/topic/530/funciones-de-hash-de-contraseña>

Capítulo 49: Galletas

Introducción

Una cookie HTTP es una pequeña porción de datos enviados desde un sitio web y almacenados en la computadora del usuario por el navegador web del usuario mientras el usuario está navegando.

Sintaxis

- `bool setcookie(string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string $domain = "" [, bool $secure = false [, bool $httponly = false]]]]]])`

Parámetros

parámetro	detalle
nombre	El nombre de la cookie. Esta es también la clave que puede utilizar para recuperar el valor de <code>\$_COOKIE</code> super global. <i>Este es el único parámetro requerido.</i>
valor	El valor para almacenar en la cookie. Esta información es accesible para el navegador, así que no almacene nada sensible aquí.
expirar	Una marca de tiempo de Unix que representa cuándo debe expirar la cookie. Si se establece en cero, la cookie caducará al final de la sesión. Si se establece en un número menor que la marca de tiempo actual de Unix, la cookie caducará inmediatamente.
camino	El alcance de la cookie. Si se establece en <code>/</code> la cookie estará disponible dentro de todo el dominio. Si se establece en <code>/some-path/</code> , la cookie solo estará disponible en esa ruta y los descendientes de esa ruta. El valor predeterminado es la ruta actual del archivo en el que se establece la cookie.
dominio	El dominio o subdominio en el que está disponible la cookie. Si se establece en el dominio pelado <code>stackoverflow.com</code> , la cookie estará disponible para ese dominio y todos los subdominios. Si se establece en un subdominio <code>meta.stackoverflow.com</code> , la cookie estará disponible solo en ese subdominio y en todos los subdominios.
seguro	Cuando se establece en <code>TRUE</code> la cookie solo se establecerá si existe una conexión segura HTTPS entre el cliente y el servidor.
httponly	Especifica que la cookie solo debe estar disponible a través del protocolo HTTP / S y no debe estar disponible para lenguajes de script del lado del cliente como JavaScript. Solo disponible en PHP 5.2 o posterior.

Observaciones

Vale la pena señalar que la mera invocación de la función `setcookie` no solo pone los datos dados en la matriz superglobal `$_COOKIE`.

Por ejemplo, no tiene sentido hacer:

```
setcookie("user", "Tom", time() + 86400, "/");
var_dump(isset($_COOKIE['user'])); // yields false or the previously set value
```

El valor aún no está allí, no hasta la próxima página de carga. La función `setcookie` simplemente dice "*con la siguiente conexión http le dice al cliente (navegador) que configure esta cookie*". Luego, cuando los encabezados se envían al navegador, contienen este encabezado de cookie. Luego, el navegador comprueba si la cookie no ha caducado, y si no, entonces, en la solicitud http, envía la cookie al servidor y ahí es cuando PHP la recibe y coloca el contenido en la matriz `$_COOKIE`.

Examples

Configuración de una cookie

Se establece una cookie utilizando la función `setcookie()`. Dado que las cookies son parte del encabezado HTTP, debe configurar las cookies antes de enviar cualquier salida al navegador.

Ejemplo:

```
setcookie("user", "Tom", time() + 86400, "/"); // check syntax for function params
```

Descripción:

- Crea una cookie con nombre de `user`
- (Opcional) El valor de la cookie es `Tom`
- (Opcional) La cookie caducará en 1 día (86400 segundos)
- (Opcional) La cookie está disponible en todo el sitio web /
- (Opcional) La cookie solo se envía a través de HTTPS
- (Opcional) La cookie no es accesible para lenguajes de script como JavaScript

Solo se puede acceder a una cookie creada o modificada en solicitudes posteriores (donde coincida la `path` y el `domain`) ya que el superglobal `$_COOKIE` no se rellena con los nuevos datos inmediatamente.

Recuperar una cookie

Recuperar y dar salida a una cookie con nombre de `user`

El valor de una cookie se puede recuperar utilizando la variable global `$_COOKIE`. Por ejemplo, si tenemos una cookie llamada `user`, podemos recuperarla de esta manera

```
echo $_COOKIE['user'];
```

Modificar una cookie

El valor de una cookie se puede modificar restableciendo la cookie

```
setcookie("user", "John", time() + 86400, "/"); // assuming there is a "user" cookie already
```

Las cookies son parte del encabezado HTTP, por `setcookie()` que se debe llamar a `setcookie()` antes de enviar cualquier salida al navegador.

Al modificar una cookie, asegúrese de que la `path` y `domain` parámetros de `domain` de `setcookie()` coincidan con la cookie existente o se creará una nueva cookie en su lugar.

La parte del valor de la cookie se codificará automáticamente cuando envíe la cookie, y cuando se reciba, se descodificará automáticamente y se asignará a una variable con el mismo nombre que el nombre de la cookie.

Comprobando si una cookie está configurada

Use la función `isset()` sobre la variable superglobal `$_COOKIE` para verificar si una cookie está configurada.

Ejemplo:

```
// PHP <7.0
if (isset($_COOKIE['user'])) {
    // true, cookie is set
    echo 'User is ' . $_COOKIE['user'];
} else {
    // false, cookie is not set
    echo 'User is not logged in';
}

// PHP 7.0+
echo 'User is ' . $_COOKIE['user'] ?? 'User is not logged in';
```

Eliminar una cookie

Para eliminar una cookie, establezca la fecha y hora de caducidad en un tiempo en el pasado. Esto activa el mecanismo de eliminación del navegador:

```
setcookie('user', '', time() - 3600, '/');
```

Al eliminar una cookie, asegúrese de que la `path` y `domain` parámetros de `domain` de `setcookie()` coincidan con la cookie que está intentando eliminar o se creará una nueva cookie que caduca de inmediato.

También es una buena idea desactivar el valor `$_COOKIE` en caso de que la página actual lo use:

```
unset($_COOKIE['user']);
```

Lea Galletas en línea: <https://riptutorial.com/es/php/topic/501/galletas>

Capítulo 50: Generadores

Examples

¿Por qué usar un generador?

Los generadores son útiles cuando necesitas generar una colección grande para iterar más tarde. Son una alternativa más simple a la creación de una clase que implementa un [iterador](#), que a menudo es una exageración.

Por ejemplo, considere la siguiente función.

```
function randomNumbers(int $length)
{
    $array = [];

    for ($i = 0; $i < $length; $i++) {
        $array[] = mt_rand(1, 10);
    }

    return $array;
}
```

Todo lo que hace esta función es generar una matriz que está llena de números aleatorios. Para usarlo, podríamos hacer `randomNumbers(10)`, lo que nos dará una matriz de 10 números aleatorios. ¿Y si queremos generar un millón de números aleatorios? `randomNumbers(1000000)` lo hará por nosotros, pero a un costo de memoria. Un millón de enteros almacenados en una matriz utiliza aproximadamente **33 megabytes** de memoria.

```
$startMemory = memory_get_usage();

$randomNumbers = randomNumbers(1000000);

echo memory_get_usage() - $startMemory, ' bytes';
```

Esto se debe a que un millón de números aleatorios completos se generan y se devuelven a la vez, en lugar de uno a la vez. Los generadores son una manera fácil de resolver este problema.

Reescribiendo números aleatorios () usando un generador

Nuestra función `randomNumbers()` puede reescribirse para usar un generador.

```
<?php

function randomNumbers(int $length)
{
    for ($i = 0; $i < $length; $i++) {
        // yield tells the PHP interpreter that this value
        // should be the one used in the current iteration.
        yield mt_rand(1, 10);
```

```

        }
    }

foreach (randomNumbers(10) as $number) {
    echo "$number\n";
}

```

Usando un generador, no tenemos que construir una lista completa de números aleatorios para regresar de la función, lo que lleva a que se use mucho menos memoria.

Leyendo un archivo grande con un generador.

Un caso de uso común para los generadores es leer un archivo del disco e iterar sobre su contenido. A continuación se muestra una clase que le permite iterar sobre un archivo CSV. El uso de memoria para este script es muy predecible y no fluctuará dependiendo del tamaño del archivo CSV.

```

<?php

class CsvReader
{
    protected $file;

    public function __construct($filePath) {
        $this->file = fopen($filePath, 'r');
    }

    public function rows()
    {
        while (!feof($this->file)) {
            $row = fgetcsv($this->file, 4096);

            yield $row;
        }
    }

    return;
}

$csv = new CsvReader('/path/to/huge/csv/file.csv');

foreach ($csv->rows() as $row) {
    // Do something with the CSV row.
}

```

La palabra clave de rendimiento

Una declaración de `yield` es similar a una declaración de retorno, excepto que en lugar de detener la ejecución de la función y devolverla, el rendimiento devuelve un objeto [Generador](#) y detiene la ejecución de la función del generador.

Aquí hay un ejemplo de la función de rango, escrita como un generador:

```
function gen_one_to_three() {
```

```
for ($i = 1; $i <= 3; $i++) {
    // Note that $i is preserved between yields.
    yield $i;
}
```

Puede ver que esta función devuelve un objeto **Generador** al inspeccionar la salida de `var_dump`:

```
var_dump(gen_one_to_three())

# Outputs:
class Generator (0) {
```

Valores de rendimiento

El objeto **Generador** se puede iterar sobre una matriz.

```
foreach (gen_one_to_three() as $value) {
    echo "$value\n";
}
```

El ejemplo anterior dará como resultado:

```
1
2
3
```

Rendimiento de valores con claves

Además de generar valores, también puede generar pares clave / valor.

```
function gen_one_to_three() {
    $keys = ["first", "second", "third"];

    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $keys[$i - 1] => $i;
    }
}

foreach (gen_one_to_three() as $key => $value) {
    echo "$key: $value\n";
}
```

El ejemplo anterior dará como resultado:

```
first: 1
second: 2
third: 3
```

Uso de la función send () para pasar valores a un generador

Los generadores están codificados rápidamente y, en muchos casos, son una alternativa delgada a las implementaciones de iteradores pesados. Con la implementación rápida viene una pequeña falta de control cuando un generador debe dejar de generar o si debe generar algo más. Sin embargo, esto se puede lograr con el uso de la función `send()`, lo que permite a la función de solicitud enviar parámetros al generador después de cada ciclo.

```
//Imagining accessing a large amount of data from a server, here is the generator for this:  
function generateDataFromServerDemo()  
{  
    $indexCurrentRun = 0; //In this example in place of data from the server, I just send  
feedback everytime a loop ran through.  
  
    $timeout = false;  
    while (!{$timeout})  
    {  
        $timeout = yield $indexCurrentRun; // Values are passed to caller. The next time the  
generator is called, it will start at this statement. If send() is used, $timeout will take  
this value.  
        $indexCurrentRun++;  
    }  
  
    yield 'X of bytes are missing. <br>';  
}  
  
// Start using the generator  
$generatorDataFromServer = generateDataFromServerDemo();  
foreach($generatorDataFromServer as $numberOfRuns)  
{  
    if ($numberOfRuns < 10)  
    {  
        echo $numberOfRuns . "<br>";  
    }  
    else  
    {  
        $generatorDataFromServer->send(true); //sending data to the generator  
        echo $generatorDataFromServer->current(); //accessing the latest element (hinting how  
many bytes are still missing.  
    }  
}
```

Resultando en esta salida:

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
X bytes are missing.
```

Lea Generadores en línea: <https://riptutorial.com/es/php/topic/1684/generadores>

Capítulo 51: Gerente de dependencia del compositor

Introducción

Composer es el gestor de dependencias más utilizado de PHP. Es análogo a `npm` en Node, `pip` para Python o `NuGet` para .NET.

Sintaxis

- ruta php / a / composer.phar [comando] [opciones] [argumentos]

Parámetros

Parámetro	Detalles
licencia	Define el tipo de licencia que desea utilizar en el Proyecto.
autores	Define los autores del proyecto, así como los detalles del autor.
apoyo	Define los correos de soporte, el canal de irc y varios enlaces.
exigir	Define las dependencias reales, así como las versiones del paquete.
require-dev	Define los paquetes necesarios para desarrollar el proyecto.
sugerir	Define las sugerencias de paquetes, es decir, paquetes que pueden ayudar si se instalan.
carga automática	Define las políticas de carga automática del proyecto.
autoload-dev	Define las políticas de carga automática para el desarrollo del proyecto.

Observaciones

La carga automática solo funcionará para las bibliotecas que especifican información de carga automática. La mayoría de las bibliotecas cumplen y se adherirán a un estándar como [PSR-0](#) o [PSR-4](#).

Enlaces Útiles

- [Packagist](#) : navega por los paquetes disponibles (que puedes instalar con Composer).

- [Documentacion oficial](#)
- [Guía oficial de introducción](#)

Pocas sugerencias

1. Deshabilita xdebug al ejecutar Composer.
2. No ejecute Composer como `root`. Los paquetes no son de confianza.

Examples

¿Qué es el compositor?

[Composer](#) es un gestor de paquetes / dependencias para PHP. Puede usarse para instalar, realizar un seguimiento y actualizar las dependencias de su proyecto. Composer también se encarga de cargar automáticamente las dependencias en las que se basa su aplicación, lo que le permite usar fácilmente la dependencia dentro de su proyecto sin preocuparse de incluirlas en la parte superior de cualquier archivo dado.

Las dependencias para su proyecto se enumeran dentro de un archivo `composer.json` que normalmente se encuentra en la raíz de su proyecto. Este archivo contiene información sobre las versiones requeridas de paquetes para producción y también para desarrollo.

Se puede encontrar un resumen completo del esquema `composer.json` en el [sitio web de Composer](#).

Este archivo se puede editar manualmente usando cualquier editor de texto o automáticamente a través de la línea de comandos a través de comandos como el `composer require <package>` o el `composer require-dev <package>`.

Para comenzar a utilizar compositor en su proyecto, deberá crear el archivo `composer.json`. Puede crearlo manualmente o simplemente ejecutar el `composer init`. Despues de ejecutar el `composer init` en su terminal, le pedirá información básica sobre su proyecto: **Nombre del paquete** (*proveedor / paquete* - por ejemplo, `laravel/laravel`), **Descripción** - *opcional*, **Autor** y alguna otra información como Estabilidad mínima, Licencia y Requisitos Paquetes.

La clave `require` en su archivo `composer.json` especifica a Composer de qué paquetes depende su proyecto. `require` toma un objeto que asigna nombres de paquetes (por ejemplo, `monolog` / `monolog`) a restricciones de versión (por ejemplo, `1.0.*`).

```
{  
    "require": {  
        "composer/composer": "1.2.*"  
    }  
}
```

Para instalar las dependencias definidas, deberá ejecutar el comando de `composer install` y, a continuación, encontrará los paquetes definidos que coincidan con la restriccion de `version` suministrada y los descargará en el directorio del `vendor`. Es una convención colocar el código de

un tercero en un directorio llamado `vendor`.

Notará que el comando de `install` también creó un archivo `composer.lock`.

Un `composer.lock` archivo es generado automáticamente por Compositor. Este archivo se usa para rastrear las versiones instaladas actualmente y el estado de sus dependencias. La ejecución de `composer install` instalará los paquetes exactamente en el estado almacenado en el archivo de bloqueo.

Autocarga con compositor

Si bien el compositor proporciona un sistema para administrar las dependencias para proyectos PHP (por ejemplo, de [Packagist](#)), también puede servir notablemente como un cargador automático, especificando dónde buscar espacios de nombres específicos o incluir archivos de funciones genéricas.

Comienza con el archivo `composer.json`:

```
{
    // ...
    "autoload": {
        "psr-4": {
            "MyVendorName\\MyProject": "src/"
        },
        "files": [
            "src/functions.php"
        ]
    },
    "autoload-dev": {
        "psr-4": {
            "MyVendorName\\MyProject\\Tests": "tests/"
        }
    }
}
```

Este código de configuración garantiza que todas las clases en el espacio de nombres `MyVendorName\MyProject` se asignen al directorio `src` y todas las clases en `MyVendorName\MyProject\Tests` al directorio de `tests` (en relación con su directorio raíz). También incluirá automáticamente el archivo `functions.php`.

Después de poner esto en su archivo `composer.json`, ejecute la `composer update` en un terminal para que el compositor actualice las dependencias, el archivo de bloqueo y genere el archivo `autoload.php`. Cuando se implementa en un entorno de producción, usaría `composer install --no-dev`. El archivo `autoload.php` se puede encontrar en el directorio del `vendor` que se debe generar en el directorio donde reside `composer.json`.

Usted debe `require` este archivo temprana en un punto de instalación en el ciclo de vida de la aplicación mediante una línea similar a la de abajo.

```
require_once __DIR__ . '/vendor/autoload.php';
```

Una vez incluido, el archivo `autoload.php` se encarga de cargar todas las dependencias que proporcionó en su archivo `composer.json`.

Algunos ejemplos de la ruta de clase a la asignación de directorios:

- `MyVendorName\MyProject\Shapes\Square` → `src/Shapes/Square.php` .
- `MyVendorName\MyProject\Tests\Shapes\Square` → `tests/Shapes/Square.php` .

Beneficios de usar Composer

Composer realiza un seguimiento de las versiones de los paquetes que ha instalado en un archivo llamado `composer.lock`, que está destinado a comprometerse con el control de versiones, de modo que cuando se clone el proyecto en el futuro, simplemente ejecutando `composer install` se descargará e instalará todas las dependencias del proyecto.

Composer se ocupa de las dependencias de PHP por proyecto. Esto facilita tener varios proyectos en una máquina que dependen de versiones separadas de un paquete PHP.

Composer rastrea las dependencias que solo están destinadas a los entornos de desarrollo.

```
composer require --dev phpunit/phpunit
```

Composer proporciona un autocargador, lo que facilita enormemente comenzar con cualquier paquete. Por ejemplo, después de instalar `Goutte` con el `composer require fabpot/goutte`, puede comenzar a usar Goutte en un nuevo proyecto de inmediato:

```
<?php  
  
require __DIR__ . '/vendor/autoload.php';  
  
$client = new Goutte\Client();  
  
// Start using Goutte
```

Composer le permite actualizar fácilmente un proyecto a la última versión permitida por su `composer.json`. P.EJ. `composer update fabpot/goutte`, o para actualizar cada una de las dependencias de su proyecto: `composer update`.

Diferencia entre "instalación del compositor" y "actualización del compositor"

```
composer update
```

`composer update` nuestras dependencias según se especifican en `composer.json`.

Por ejemplo, si nuestro proyecto usa esta configuración:

```
"require": {  
    "laravelcollective/html": "2.0.*"  
}
```

Suponiendo que hayamos instalado la versión 2.0.1 del paquete, la `composer update` ejecución provocará una actualización de este paquete (por ejemplo, a 2.0.2, si ya se ha publicado).

En detalle la `composer update`:

- Leer `composer.json`
- Elimine los paquetes instalados que ya no sean necesarios en `composer.json`
- Compruebe la disponibilidad de las últimas versiones de nuestros paquetes requeridos.
- Instala las últimas versiones de nuestros paquetes.
- Actualice `composer.lock` para almacenar la versión de los paquetes instalados.

`composer install`

`composer install` todas las dependencias especificadas en el archivo `composer.lock` en la versión especificada (bloqueada), sin actualizar nada.

En detalle:

- Leer el archivo `composer.lock`
- Instala los paquetes especificados en el archivo `composer.lock`.

Cuándo instalar y cuándo actualizar.

- `composer update` se utiliza principalmente en la fase de 'desarrollo', para actualizar nuestros paquetes de proyectos.
- `composer install` se utiliza principalmente en la 'fase de implementación' para instalar nuestra aplicación en un servidor de producción o en un entorno de prueba, utilizando las mismas dependencias almacenadas en el archivo `composer.lock` creado por la `composer update`.

Composer de comandos disponibles

Mando	Uso
acerca de	Breve información sobre el compositor
archivo	Crear un archivo de este paquete compositor
vistazo	Abre la URL del repositorio del paquete o la página de inicio en su navegador.
limpiar cache	Borra la caché interna del paquete del compositor.
limpiar cache	Borra la caché interna del paquete del compositor.
configuración	Establecer opciones de configuración
crear proyecto	Crear nuevo proyecto desde un paquete en el directorio dado.

Mando	Uso
depende	Muestra qué paquetes hacen que se instale el paquete dado
diagnosticar	Diagnostica el sistema para identificar errores comunes.
volcado-autoload	Vuelca el autoloader
Dumpautoload	Vuelca el autoloader
exec	Ejecutar un script binario / vendedor
global	Permite ejecutar comandos en el directorio del compositor global (\$COMPOSER_HOME).
ayuda	Muestra ayuda para un comando
casa	Abre la URL del repositorio del paquete o la página de inicio en su navegador.
info	Mostrar información sobre paquetes
en eso	Crea un archivo composer.json básico en el directorio actual.
instalar	Instala las dependencias del proyecto desde el archivo composer.lock, si está presente, o recae en el composer.json.
licencias	Mostrar información sobre licencias de dependencias.
lista	Listas de comandos
antiquado	Muestra una lista de paquetes instalados que tienen actualizaciones disponibles, incluida su última versión.
prohíbe	Muestra qué paquetes impiden que se instale el paquete dado
retirar	Elimina un paquete del require o require-dev
exigir	Agrega los paquetes requeridos a tu composer.json y los instala
ejecutar guión	Ejecute los scripts definidos en composer.json.
buscar	Buscar paquetes
auto-actualización	Actualiza composer.phar a la última versión.
auto actualización	Actualiza composer.phar a la última versión.

Mando	Uso
espectáculo	Mostrar información sobre paquetes
estado	Mostrar una lista de paquetes modificados localmente
sugiere	Mostrar sugerencias de paquetes
actualizar	Actualiza tus dependencias a la última versión de acuerdo con composer.json y actualiza el archivo composer.lock.
validar	Valida un composer.json y composer.lock
por qué	Muestra qué paquetes hacen que se instale el paquete dado
Por qué no	Muestra qué paquetes impiden que se instale el paquete dado

Instalación

Puede instalar Composer localmente, como parte de su proyecto, o globalmente como un ejecutable de todo el sistema.

En la zona

Para instalar, ejecute estos comandos en su terminal.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
# to check the validity of the downloaded installer, check here against the SHA-384:
# https://composer.github.io/pubkeys.html
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Esto descargará `composer.phar` (un archivo de archivo PHP) al directorio actual. Ahora puede ejecutar `php composer.phar` para usar Composer, por ejemplo

```
php composer.phar install
```

Globalmente

Para usar Composer globalmente, coloque el archivo `composer.phar` en un directorio que sea parte de su `PATH`

```
mv composer.phar /usr/local/bin/composer
```

Ahora puedes usar `composer` cualquier lugar en lugar de `php composer.phar`, por ejemplo

```
composer install
```

Lea Gerente de dependencia del compositor en línea:

<https://riptutorial.com/es/php/topic/1053/gerente-de-dependencia-del-compositor>

Capítulo 52: Imaginario

Examples

Primeros pasos

Instalación

Usando apt en sistemas basados en Debian

```
sudo apt-get install php5-imagick
```

Usando Homebrew en OSX / macOs

```
brew install imagemagick
```

Para ver las dependencias instaladas usando el método `brew`, visite brewformulas.org/Imagemagick.

Usando lanzamientos binarios

Instrucciones en el [sitio web de imagemagick](#).

Uso

```
<?php  
  
$imagen = new Imagick('imagen.jpg');  
$imagen->thumbnailImage(100, 0);  
//if you put 0 in the parameter aspect ratio is maintained  
  
echo $imagen;  
  
?>
```

Convertir imagen en base64 String

Este ejemplo es cómo convertir una imagen en una cadena Base64 (es decir, una cadena que puede usar directamente en un atributo `src` de una etiqueta `img`). Este ejemplo utiliza específicamente la biblioteca `Imagick` (también hay otras disponibles, como `GD`).

```
<?php  
/**  
 * This loads in the file, image.jpg for manipulation.  
 * The filename path is relative to the .php file containing this code, so  
 * in this example, image.jpg should live in the same directory as our script.  
 */  
$img = new Imagick('image.jpg');
```

```

/**
 * This resizes the image, to the given size in the form of width, height.
 * If you want to change the resolution of the image, rather than the size
 * then $img->resampleimage(320, 240) would be the right function to use.
 *
 * Note that for the second parameter, you can set it to 0 to maintain the
 * aspect ratio of the original image.
 */
$img->resizeImage(320, 240);

/**
 * This returns the unencoded string representation of the image
 */
$imgBuff = $img->getimageblob();

/**
 * This clears the image.jpg resource from our $img object and destroys the
 * object. Thus, freeing the system resources allocated for doing our image
 * manipulation.
 */
$img->clear();

/**
 * This creates the base64 encoded version of our unencoded string from
 * earlier. It is then output as an image to the page.
 *
 * Note, that in the src attribute, the image/jpeg part may change based on
 * the image type you're using (i.e. png, jpg etc).
 */
$img = base64_encode($imgBuff);
echo "<img alt='Embedded Image' src='data:image/jpeg;base64,$img' />";

```

Lea Imaginario en línea: <https://riptutorial.com/es/php/topic/7682/imaginario>