

# PHP – ARRAYS

Um array (ou vetor) é uma coleção de variáveis indexadas e armazenadas numa única variável

Esta variável, facilmente referenciada, oferece uma maneira fácil de passar múltiplos valores entre linhas de código, funções ou até páginas

Muitas das funções de ambiente PHP estão na forma de arrays (p.e. `$_SESSION`)

Qualquer situação que envolva várias porções de dados pode ser condensado e gerido mais apropriado num array PHP

# PHP - ARRAYS

Os arrays PHP são associativos com pouco processamento extra

A parte associativa significa que o array armazena valores de elementos associados a uma chave, ao invés de associar um índice estritamente inteiro

Quem tem experiência com arrays noutras linguagens de programação consegue perceber a diferença

# PHP - ARRAYS

Se armazenarmos um elemento num array, associado a uma chave, apenas precisamos da chave para aceder ao valor mais tarde

Por exemplo, podemos armazenar simplesmente assim:

```
$state_location['San Mateo'] = 'California';
```

Armazena o elemento 'California' na variável array \$state\_location, em associação com a chave 'San Mateo'

Depois de ter sido armazenada, podemos obter o valor usando a chave:

```
$state = $state_location['San Mateo']; // vamos obter 'California'
```

# PHP - ARRAYS

Se apenas precisarmos de arrays para armazenar valores/chaves, a informação anterior é suficiente

Da mesma forma, se quisermos associar uma ordem numérica a um conjunto de valores, apenas temos que usar índices inteiros da forma convencional:

```
$my_array[1] = “The first thing”;
```

```
$my_array[2] = “The second thing”; // e assim sucessivamente
```

# PHP - ARRAYS

Em adição ao processamento extra que torna este tipo de associação chave/valor possível, os arrays trazem também outras adições

Por esta razão, por vezes tratamo-los como outro tipo de estruturas de dados

Os arrays podem ser *multidimensionais*

Podem armazenar valores associados a uma sequência de chaves de valores ao invés de uma única chave

Os arrays também podem manter uma lista ordenada de elementos que tenha sido inserida, independentemente das chaves dos valores inseridos

Isto torna possível tratar arrays como listas ligadas

# PHP - ARRAYS

Ao contrário de outras linguagens de programação, Como C, os arrays PHP são associativos

Ao invés de terem um número fixo de espaços em memória, o PHP cria os espaços à medida que novos elementos são adicionados ao array

Ao invés de requerer elementos do mesmo tipo, os arrays PHP possuem a mesma característica sem tipo que as variáveis PHP têm – podemos atribuir valores arbitrários de tipos diferentes aos elementos do array

Por fim, e porque os vetores de arrays são na sua definição elementos na sua posição numérica, as chaves para pesquisa e obtenção de valor devem ser inteiros

# PHP - ARRAYS

A maneira mais simples de criar um array é assumir que a variável já é um array e atribuir um valor do tipo :

```
$my_array[1] = "The first thing in my array that I just made";
```

Se o `$my_array` não estava definido antes, vai ser agora uma variável de array com um elemento

Se ao invés de a variável `$my_array` já fosse um array, a string vai ser associada à chave 1

Se não existir um valor associado àquele número, um novo espaço vai ser alocado para ele

Se um valor já estivesse associado à chave 1, o valor vai ser escrito por cima

# PHP - ARRAYS

Outra forma de criar um array é pelo método `array()`, que cria um novo array com a especificação dada dos elementos e chaves associadas

Na sua forma mais simples, `array()` é chamada sem argumentos, criando um array vazio

Outra maneira simples de chamar a função, é introduzir uma lista separada por vírgulas de elementos a serem armazenados, sem especificar chaves



# PHP - ARRAYS

○ resultado é que os elementos são armazenados no array na ordem especificada com índices inteiros a começar em 0

Por exemplo a declaração:

```
$fruit_basket = array('apple', 'orange', 'banana', 'pear');
```

Vai associar à variável `$fruit_basket` um array com quatro elementos de texto ('apple', 'banana', 'orange', 'pear'), com os índices 0, 1, 2, e 3, respetivamente

○ array vai lembrar a ordem pelo qual os elementos foram armazenados

# PHP - ARRAYS

A atribuição a `$fruit_basket` tem então o mesmo efeito do que o seguinte:

- `$fruit_basket[0] = 'apple';`
- `$fruit_basket[1] = 'orange';`
- `$fruit_basket[2] = 'banana';`
- `$fruit_basket[3] = 'pear';`

# PHP - ARRAYS

Esta expressão também vai ter o mesmo resultado:

```
$fruit_basket = array(0 => 'apple', 1 => 'orange',  
2 => 'banana', 3 => 'pear');
```

Mas podemos utilizar outras chaves para indexar os valores:

```
$fruit_basket = array('red' => 'apple', 'orange' => 'orange',  
'yellow' => 'banana', 'green' => 'pear');
```

# PHP - ARRAYS

Outra forma de criar um array é chamar uma função que retorna um array

Isto pode uma função do utilizador ou uma função nativa do PHP

Muitas funções de bases de dados, por exemplo, retornam os seus resultados em arrays

Outras funções existem simplesmente para criar arrays para poderem ser úteis e mais tarde manipulados de alguma forma

# PHP - ARRAYS

Uma dessas funções é a `range()`, que toma dois inteiros como argumentos e retorna um array preenchido com todos os inteiros entre os argumentos (inclusive)

Por outras palavras:

```
$my_array = range(1,5);
```

é equivalente a:

```
$my_array = array(1, 2, 3, 4, 5);
```

# PHP - ARRAYS

A forma mais direta de obter um valor é o seu índice

Se temos armazenado um valor no índice 5 do array `$my_array`, `$my_array[5]` deve devolver o valor em causa

Se o `$my_array` nunca foi declarado, o `$my_array[5]` vai comportar-se como uma variável não existente

# PHP - ARRAYS

Há mais formas de obter valores de arrays sem chaves, a maior parte explora o facto de os arrays guardarem silenciosamente a ordem dos elementos

As iterações são usadas para obter vários elementos do array em sucessão

Suponhamos que temos as seguintes linhas executadas:

```
$fruit_basket = array('apple', 'orange', 'banana');  
list($red_fruit, $orange_fruit) = $fruit_basket;
```

# PHP - ARRAYS

Isto irá atribuir o texto 'apple' à variável \$red\_fruit e o texto 'orange' à variável \$orange\_fruit

Nenhuma atribuição é feita a 'banana', porque não fornecemos variáveis suficientes

As variáveis em list() vão ser atribuídas aos elementos do array na ordem em que foram originalmente armazenadas no array

O construtor de list() está do lado esquerdo do operador (=), onde normalmente encontramos variáveis



# PHP - ARRAYS

De alguma forma, `list()` é o oposto ou inverso de `array()` porque a função `array()` compacta os seus argumentos num array e `list()` toma o array e reparte-o em atribuições individuais de variáveis

Se avaliarmos:

```
list($first, $second) = array($first, second);
```

Os valores originais de `$first` e `$second` vão ser atribuídos a essas variáveis novamente, após terem sido temporariamente armazenadas num array

# PHP - ARRAYS

O PHP pode facilmente suportar arrays multidimensionais, com números e tipos de chaves de vários tipos

Não existe necessidade de o declarar, uma variável array pode ser uma atribuição do tipo :

```
$multi_array[1][2][3][4][5] = “deeply buried treasure”;
```

Isto é um array de 5 dimensões com chaves sucessivas, que neste caso são inteiros consecutivos

# PHP - ARRAYS

Esta sintaxe de índice múltiplo no exemplo anterior é simplesmente uma maneira sucinta de nos referirmos a um array multidimensional de dimensão 4 que está armazenado na chave `l` do `$multi_array`, que por sua vez tem um array (tridimensional) armazenado em `si`, e assim sucessivamente

É preciso notar que podemos ter diferentes profundidades em diferentes partes do array tais como:

```
$multi_level_array[0] = "a simple string";
```

```
$multi_level_array[1]['contains'] = "a string stored deeper";
```

A chave inteira `0` armazena um texto, e a chave `l` armazena um array que por sua vez tem um texto dentro dela

# PHP - ARRAYS

No entanto, não podemos continuar com esta atribuição:

```
$multi_level_array[0]['contains'] = "another deep string"; ❌
```

Sem ter como consequência a perda da primeira atribuição a 'a simple string'

A chave 0 pode ser usada para armazenar um texto ou outro array, mas não pode armazenar os dois em simultâneo

# PHP - ARRAYS

Exemplo:

```
$cornucopia = array('fruit' => array('red' => 'apple',  
                                     'orange' => 'orange',  
                                     'yellow' => 'banana',  
                                     'green' => 'pear'),  
                   'flower' =>  
                   array('red' => 'rose',  
                         'yellow' => 'sunflower',  
                         'purple' => 'iris'));
```

Um simples array com dois valores associados a chaves

# PHP - ARRAYS

Cada um destes valores é um array em si

Podem ser referenciados como:

```
$kind_wanted = 'flower';
```

```
$color_wanted = 'purple';
```

```
print("The $color_wanted $kind_wanted is " .
```

```
$cornucopia[$kind_wanted][$color_wanted]);
```

O resultado no navegador vai ser:

The purple flower is iris

# PHP - ARRAYS

Não existe penalização por chamar o índice errado num array multidimensional  
Se a chave não for encontrada, a expressão é tratada como uma variável não declarada

O código seguinte:

```
$kind_wanted = 'fruit';  
$color_wanted = 'purple'; // não armazenamos isto  
print("The $color_wanted $kind_wanted is " .  
$cornucopia[$kind_wanted][$color_wanted]);
```

Vai resultar em:

The purple fruit is

# PHP - ARRAYS

Função	Comportamento
<code>is_array()</code>	Takes a single argument of any type and returns a true value if the argument is an array, and false otherwise
<code>count()</code>	Takes an array as argument and returns the number of nonempty elements in the array
<code>sizeof()</code>	Identical to <code>count()</code>
<code>in_array()</code>	Takes two arguments: an element and an array. Returns true if the element is contained as a value in the array, false otherwise
<code>isset(\$array[\$key])</code>	



# PHP - ARRAYS

Apagar um elemento de um array é simples, basta chamar a função unset:

```
$my_array[0] = 'wanted';  
$my_array[1] = 'unwanted';  
$my_array[2] = 'wanted again';  
unset($my_array[1]);
```

Assumindo que o `$my_array` não estava declarado quando começámos, no fim vamos ter dois valores

(`'wanted'`, `'wanted again'`)

# PHP - ARRAYS

Não é o mesmo do que colocar o conteúdo vazio

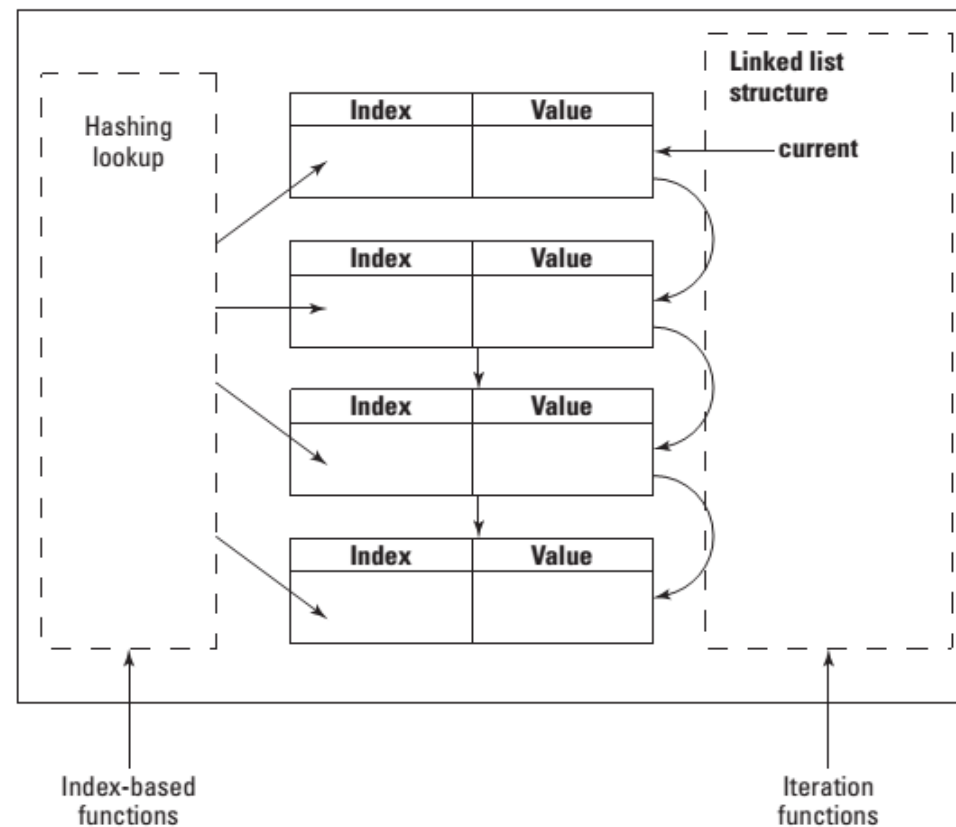
Se afirmarmos:

```
$my_array[] = “;
```

No final vamos ter na mesma 3 valores armazenados  
(‘wanted’, ‘’, ‘wanted again’)

# PHP - ARRAYS

## Estrutura interna de um array



# PHP - ARRAYS

Para explorar as funções de iteração, vamos declarar o seguinte array :

```
$major_city_info = array();  
$major_city_info[0] = 'Chicago';  
$major_city_info['Chicago'] = 'United States';  
$major_city_info[1] = 'Stockholm';  
$major_city_info['Stockholm'] = 'Sweden';  
$major_city_info[2] = 'Montreal';  
$major_city_info['Montreal'] = 'Canada'
```

# PHP - ARRAYS

Podemos escrever uma função que imprime a cidade e o país a que pertence:

```
function city_by_number ($number_index, $city_array)
{
    if (IsSet($city_array[$number_index]))
    {
        $the_city = $city_array[$number_index];
        $the_country = $city_array[$the_city];
        print("$the_city is in $the_country<BR>");
    }
}
city_by_number(0, $major_city_info);
city_by_number(1, $major_city_info);
city_by_number(2, $major_city_info);
```

## PHP - ARRAYS

Com o bloco anterior de código, a saída vai ser:

Chicago is in United States

Stockholm is in Sweden

Montreal is in Canada

# PHP - ARRAYS

Um dos iteradores mais interessantes é o foreach

```
foreach ($array_variable as $value_variable) {  
    // .. Fazer alguma coisa com o valor de $value_variable  
}
```

```
foreach ($array_variable as $key_var => $value_var) {  
    // .. Fazer alguma coisa com a chave $key_var e/ou o valor $value_var  
}
```

# PHP - ARRAYS

Agora podemos escrever uma função para imprimir todos os nomes do nosso array:

```
function print_all_foreach ($city_array)
{
    foreach ($city_array as $name_value) {
        print("$name_value<BR>");
    }
}
print_all_foreach($major_city_info);
```



# PHP - ARRAYS

Podemos também obter o conteúdo do array com as funções de iteração `current()` e `next()`

```
function print_all_next($city_array)
{
    $current_item = current($city_array);
    if ($current_item)
        print("$current_item<BR>");
    else
        print("Nada para imprimir");
    while($current_item = next($city_array))
        print("$current_item<BR>");
}
print_all_next($major_city_info);
```

# PHP - ARRAYS

Da mesma forma podemos utilizar as funções `end()` and `prev()` para imprimir as nossas entradas por ordem inversa

```
function print_all_array_backwards($city_array)
{
    $current_item = end($city_array); // Avançar para o último
    if ($current_item)
        print("$current_item<BR>");
    else
        print("Nada a imprimir");
    while($current_item = prev($city_array))
        print("$current_item<BR>");
}
print_all_array_backwards($major_city_info);
```

# PHP - ARRAYS

Utilizando a função `key()`, podemos modificar uma das funções anteriores para imprimir chaves e valores simultaneamente

```
function print_keys_and_values($city_array) {  
    reset($city_array);  
    $current_value = current($city_array);  
    $current_key = key($city_array);  
    if ($current_value)  
        print("Chave: $current_key; Valor: $current_value<BR>");  
    else  
        print("Nada a imprimir");  
    while($current_value = next($city_array))  
    {  
        $current_key = key($city_array);  
        print("Key: $current_key; Value: $current_value<BR>");  
    }  
}  
print_keys_and_values($major_city_info);
```

# PHP - ARRAYS

Isto vai-nos dar o seguinte resultado:

Chave: 0; Valor: Chicago

Chave : Chicago; Valor : United States

Chave : 1; Valor : Stockholm

Chave : Stockholm; Valor : Sweden

Chave : 2; Valor : Montreal

Chave : Montreal; Valor: Canada

# PHP - ARRAYS

Podemos usar também a função `each()` para escrever uma versão mais robusta e imprimir todas as chaves e valores:

```
function print_keys_and_values_each($city_array)
{
    reset($city_array);
    while ($array_cell = each($city_array))
    {
        $current_value = $array_cell['value'];
        $current_key = $array_cell['key'];
        print("Chave: $current_key; Valor: $current_value<BR>");
    }
}
print_keys_and_values_each($major_city_info);
```

# PHP - ARRAYS

Aplicando esta função vamos ter como resultado :

Chave: 0; Valor: Chicago

Chave : Chicago; Valor : United States

Chave : 1; Valor : Stockholm

Chave : Stockholm; Valor : Sweden

Chave : 2; Valor : Montreal

Chave : Montreal; Valor: Canada

# PHP - ARRAYS

Se passarmos o array simples usando a função `array_walk()`:

```
array_walk($major_city_info, 'print_value_length');
```

○ resultado vai ser:

The length of Chicago is 7

The length of United States is 13

The length of Stockholm is 9

The length of Sweden is 6

The length of Montreal is 8

The length of Canada is 6

# PHP - ARRAYS

Função	Argumentos	Efeito secundário	Valor de retorno
current()	One array argument	None	The value from the key/value pair currently pointed
next()	One array argument	Advances the pointer by one [...]	The value pointed to after it has been advanced
prev()	One array argument	Moves the pointer back by one	The value pointed to after it has been moved back
reset()	One array argument	Moves the pointer back to point to the first key	The first value stored in the array
end()	One array argument		



# PHP - ARRAYS

Função	Argumentos	Efeito secundário	Valor de retorno
<code>pos()</code>	One array argument	Alias for <code>current()</code>	The value currently pointed
<code>each()</code>	One array argument	Moves the pointer ahead to the next key/value pair	An array that packages the keys and values of the key/value pair that was current before the pointer was moved [...]
<code>array_walk()</code>	1) An array argument 2) The name argument function to call 3) an optional argument	This function invokes the function named by its second argument on each key/value pair [...]	Returns 1

# PHP - ARRAYS

## Outras funções de arrays

### Pesquisa de dados:

`bool in_array ( mixed $needle , array $haystack [, bool $strict = FALSE ] )`

`mixed array_search ( mixed $needle , array $haystack [, bool $strict = false ] )`

`bool array_key_exists ( mixed $key , array $array )`

# PHP - ARRAYS

## Outras funções de arrays

### Ordenação de dados:

`bool asort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`

`bool ksort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`

# PHP - ARRAYS

Outras funções de arrays

Arrays aleatórios:

bool shuffle ( **array** &\$array )

mixed array\_rand ( **array** \$array [, int \$num = 1 ] )

Outras funções na documentação

<http://php.net/manual/en/ref.array.php>

# PHP – COOKIES E SESSÕES

Uma sessão de navegação é um período de tempo durante o qual uma pessoa visualiza um número de páginas e depois termina

A sessão decorre desde o primeiro pedido do sítio até o fecho do navegador ou visita a outro sítio

# PHP - COOKIES E SESSÕES

Porquê?

Queremos personalizar a experiência dos utilizadores à medida que navegam no sítio

De uma forma que dependa das páginas que já visitaram

Queremos mostrar publicidade ao utilizador

Mas não queremos mostrar os mesmos mais do que uma vez

Queremos que a sessão armazena informação acerca das ações do utilizador

Como num jogo com pontos ou num carrinho de compras num sítio de compras

Estamos interessado em como as pessoas navegam através do sítio

Quanto chegam a uma página interna, é porque a marcaram como favorito ou for a passo-a-passo através da inicial?

# PHP - COOKIES E SESSÕES

## Como trabalham as sessões PHP

Seguimento da sessão (isto é, detetando se duas chamadas diferentes são de fato parte do mesmo utilizador)

Armazenando informação em associação com a sessão

# PHP - COOKIES E SESSÕES

Fazendo o PHP reconhecer a nossa sessão

O primeiro passo em código para usar a funcionalidade da sessão é permitir ao PHP saber que a sessão está em curso para ele poder proceder e associar informação

Isto é feito pela chamada da função `session_start()`, sem argumentos

Também se pode chamar a função `session_register()`, que causa uma chamada implícita a `session_start()`

O efeito de `session_start()` depende se o PHP consegue localizar o identificador anterior de uma sessão, tal como fornecido pelos argumentos HTTP ou num cookie

Se algum for encontrado, os valores anteriormente registados são recuperados

Se nenhum for encontrado, o PHP assume que é a primeira página na sessão e gera um novo identificador de sessão



# PHP - COOKIES E SESSÕES

## Propagar variáveis de sessão

Assumindo que se fez a chamada para a função `session_start()`, basta utilizar o array superglobal `$_SESSION` para guardar e obter informação mais tarde

```
<?php
session_start();
$temporary_number = 45;
$save_this_one = 19;
$another_temporary = 33;
$_SESSION['save_this'] = $save_this_one;
?>
```

# PHP - COOKIES E SESSÕES

Propagar variáveis de sessão

O código para obter pode ser tão simples como o exemplo:

```
<?php  
session_start();  
$saved_from_prev_page = $_SESSION['save_this'];  
[..]  
$temporary_number = 45;  
$another_temporary = 33;  
[..]  
?>
```

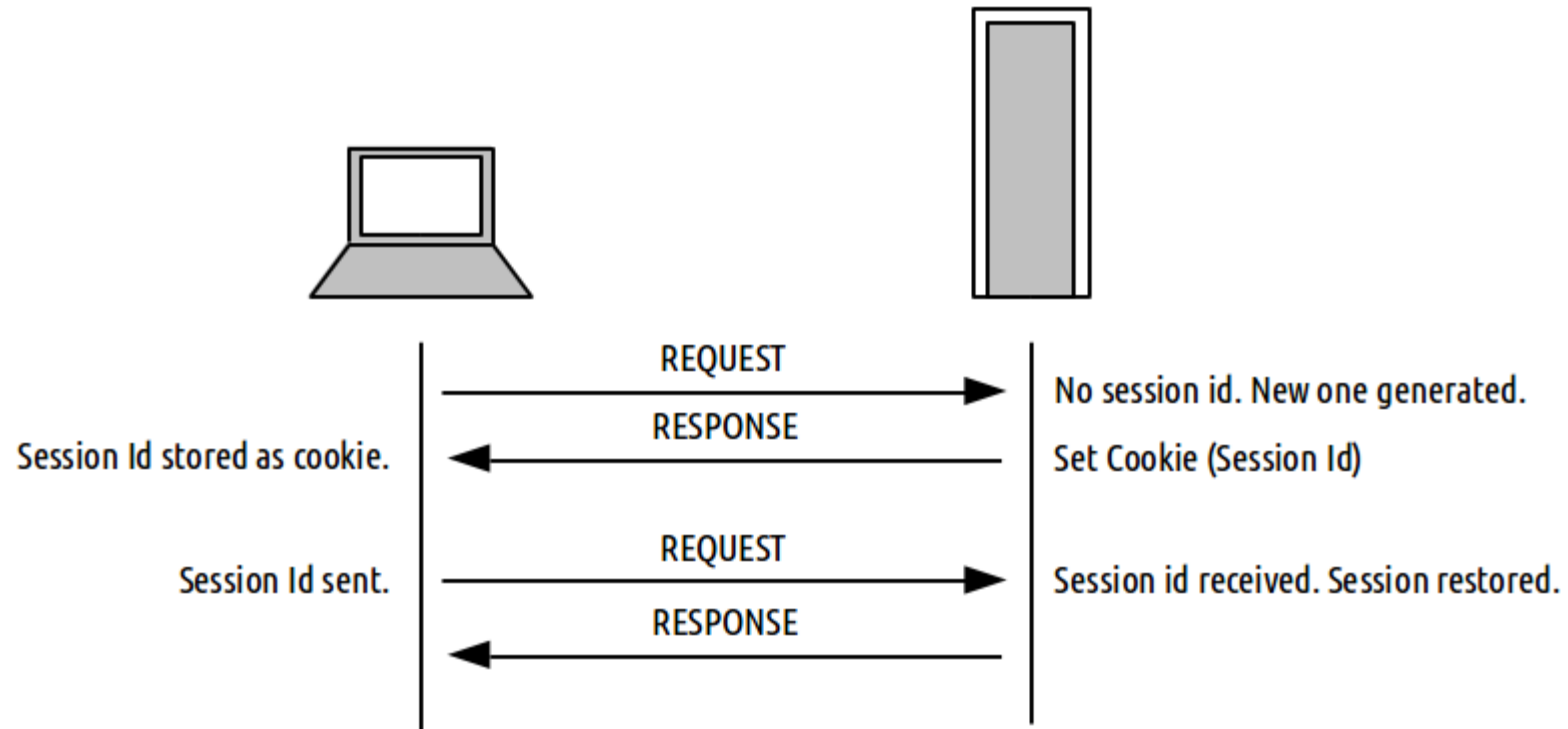
# PHP - COOKIES E SESSÕES

Um cookie é uma peça pequena de informação que é retida na máquina do utilizador, ou na memória do navegador, ou num pequeno ficheiro no disco do utilizador

Contém um par nome/valor que atribui um valor associado a uma chave no lado do cliente

Ler um cookie significa utilizar o nome para obter o valor

# PHP - COOKIES E SESSÕES



# PHP - COOKIES E SESSÕES

Normalmente apenas se guarda informação de um cookie no cliente

Armazená-lo no servidor não é uma opção viável

Em parte existe uma política de cortesia na aceitação

Tente aumentar a segurança e aceitar cookie manualmente durante uma semana e verá vários abusos neste técnica

# PHP - COOKIES E SESSÕES

Os cookies são configurados usando a função `setcookie()`, e são lidos praticamente de forma automática

Desde a versão PHP4.1, os nomes e valores dos cookies aparecem na variável superglobal `$_COOKIE`

Com o nome do cookie como índice e o valor como o valor que ele indexa

# PHP - COOKIES E SESSÕES

## Argumentos da função setcookie()

Nome do argumento	Tipo	Significado
name	string	The name of your cookie
value	string	The value you want to store
expire	int	Specifies when it expires
path	string	Sets the path to a subdirectory
httponly	boolean	Only sent through HTTP requests
domain	string	Check if domains match
secure	boolean	Sent over a secure socket if true

# PHP - COOKIES E SESSÕES

Exemplo:

```
setcookie('membername', 'timboy');
```

Isto configura um cookie chamado membername, com um valor de timboy

Este cookie vai persistir apenas até o navegador ser fechado

Vai ser lido nos pedido subsequentes neste navegador a este servidor, independentemente do domínio do pedido

Ou a partir de onde a raiz do ficheiro web for providenciado

O cookie irá ser lido independentemente de a ligação ser segura



# PHP - COOKIES E SESSÕES

Outro exemplo:

```
setcookie('membername', 'troutgirl', time() + (60 * 60 * 24), "/", "www.troutworks.com", 1);
```

Isto irá configurar o cookie para o valor 'troutgirl', substituindo o último exemplo

- tempo de validade será 86,400 segundos (1 dia) após o tempo atual

- caminho dado é o mais inclusive possível ("/")

Este cookie vai ser lido independentemente da sua localização

- argumento do anfitrião está definido como 'www.troutworks.com'

- que significa que as visualizações subsequentes não irão ler o cookie a não ser que sejam feitos a este domínio

Por fim, o último argumento especifica que este cookie será lido ou escrito por uma ligação segura

# PHP - COOKIES E SESSÕES

## Leitura cookies

Os cookies que foram definidos com sucesso num navegador serão automaticamente lidos no próximo pedido do navegador

O nome/valor do cookie será adicionado à variável `$_COOKIE`, como se tivéssemos feito a atribuição `$_COOKIE['name'] = value`

Se a diretiva `register_globals`, de versões mais antigas de PHP, estiver ativada, a variável estará disponível com o nome do cookie

# PHP - COOKIES E SESSÕES

Exemplo:

```
setcookie('membername', 'timboy');
```

O acesso na página irá ser:

```
$membername = $_COOKIE['membername'];  
print("The member name is $membername<BR>");
```

Se o register\_globals estiver ativado:

```
print("The member name is $membername<BR>");
```

# PHP - COOKIES E SESSÕES

Se definirmos um cookie num dado código, não será definido no cliente até essa página ser enviada ao mesmo

○ que será tarde de mais para ser utilizado nesse mesmo código

Isto significa que a variável global não estará disponível até fazermos o próximo pedido

# PHP - COOKIES E SESSÕES

O código seguinte não funciona da maneira como queremos:

```
setcookie('membername', 'timboy');  
print("I set a cookie! Now I will grab the value<BR>");  
// (Errado- membername vai estar vazio)  
$membername = $_COOKIE['membername'];  
print("The member name is $membername<BR>");
```

# PHP - COOKIES E SESSÕES

O código seguinte consegue contornar isso :

```
$cookievalue = 'timboy';  
setcookie('membername', $cookievalue);  
print("I set a cookie for the benefit of future pages<BR>");  
// (CERTO)  
print("Its name is membername, its value is $cookievalue<BR>");
```

# PHP - EXCEÇÕES

Exceções são eventos que podem ocorrer durante a execução de um código interrompe o fluxo normal das operações

Como noutras linguagens de programação, as exceções podem acauteladas e geridas em PHP

Para acautelar uma exceção, podemos usar o seguinte excerto:

```
if ($db == null)  
    throw new Exception('Database not initialized');
```

# PHP - EXCEÇÕES

Exception é uma classe com os seguintes métodos públicos:

```
final public string getMessage ();  
final public Exception getPrevious ();  
final public mixed getCode ();  
final public string getFile ();  
final public int getLine ();  
final public array getTrace ();  
final public string getTraceAsString ();
```



# PHP - EXCEÇÕES

O bloco try-catch consiste em:

Um bloco try

Seguido de uma ou mais cláusulas catch, que especificam como gerir diferentes exceções

```
try {  
    $car = getCar($id);  
}  
catch (DatabaseException $e) {  
    echo 'Database error: ' . $e->getMessage();  
}  
catch (Exception $e) {  
    echo 'Unknown error: ' . $e->getMessage();  
}
```

# PHP - MYSQL

O MySQL é um sistema de gestão de base de dados relacional feito em código aberto (open-source)

Tem várias versões com mais funcionalidades e suporte, com custos associados

Suporta um grande número de transações e dados

# PHP - MYSQL

○ MySQL tem um sistema privilegiado de acesso e registo

○ acesso às bases de dados ou às tabelas de uma base de dados é um privilégio limitado

É uma sistema apropriado para armazenar informação sensível, devido aos mecanismos de segurança e encriptação disponíveis

A primeira versão foi lançada 1995 e a 5.0 em 2005

Em Setembro de 2016 foi lançada a versão 8.0

# PHP – INTEGRAÇÃO DE BASES DE DADOS

A extensão MySQLi define uma interface para aceder a bases de dados  
Para nos ligarmos a uma base de dados podemos utilizar um objeto MySQLi ou o modo procedural

```
// Objeto
```

```
$sqlldb = new mysqli($servername, $username, $password);  
if ($sqlldb->connect_error) (...)
```

```
// Procedimental
```

```
$sqlldb = mysqli_connect($servername, $username, $password);  
if (!$sqlldb) (...)
```

# PHP – INTEGRAÇÃO DE BASES DE DADOS

A extensão PHP Data Objects (PDO) define uma interface para aceder a bases de dados

Para nos ligarmos a uma base de dados podemos utilizar um objeto PDO

O texto da ligação depende da BD

```
$sqldb = new PDO('mysql:host=localhost;dbname=test', $user, $pass);  
$sqldb = new PDO('pgsql:host=localhost;port=5432;dbname=anydb', $user, $pass);  
$sqldb = new PDO('sqlite:database.db');
```

## Criar Bases de dados MySQL

As BD são criadas com a chamada simples da função **mysqli\_query()** function, tal como:

```
mysqli_query($sqldb, "CREATE DATABASE myDB");
```

Em alternativa, uma interface orientada ao objeto está também disponível:

```
$sqldb->query("CREATE DATABASE myDB");
```

## Executar comandos MySQL

No ambiente, os comandos, incluindo a maior parte das interações com a BD como operações SELECT, INSERT, UPDATE e DELETE podem ser executados pelo método **mysqli\_query()**

Por exemplo:

```
$result=mysqli_query($sqlldb, "SELECT * FROM foo");
```

O método **mysqli\_query()** é utilizado então para criar tabelas e inserir dados, além de obter dados

# PHP - PDO

## Criar tabelas MySQL

O MySQL armazena os dados com domínios concretos

As colunas são auto-incrementadas, como o identificadores utilizados para chaves primárias

Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. <b>Note:</b> You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. <b>Tip:</b> You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to - 1 and 0). <b>Note:</b> Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	



## Criar tabelas MySQL

Um exemplo que cria uma tabela com os estados dos EUA:

```
mysqli_query($sqlldb, "CREATE TABLE mytable (id INT PRIMARY KEY, stateabb  
VARCHAR(12), state VARCHAR(50))");
```

A criação de tabelas apenas necessita de ser feita uma vez

A tentativa de criar uma tabela que já existe causa um aviso

## Inserção de dados MySQL

Tais como outras declarações, a inserção de dados numa é conseguida com o método **mysqli\_query()**

Poderíamos inserir alguns valores com as seguintes linhas:

```
mysqli_query($sqlldb, "INSERT INTO mytable (stateabb,state)  
VALUES ('WI','Wisconsin')");
```


```
mysqli_query($sqlldb, "INSERT INTO mytable (stateabb,state)  
VALUES ('CA','California')");
```

# PHP - PDO

## Inserção de dados MySQL

O código pré-preparado é a forma recomendada para executar comandos e prevenir ataques de SQL injection

```
$stateabb='WI';  
$state='Wisconsin';  
$result = $sqldb->prepare("INSERT INTO State (stateabb, state)  
VALUES (?, ?)");  
$result->bind_param("ss", $state, $stateabb);  
$result->execute();
```



Character	Description
i	integer
d	double
s	string
b	variable

## Obtenção de dados MySQL

Existem várias formas pelas quais podemos obter dados das tabelas da fonte de dados MySQL

O método mais simples é usando o método `fetch()`

O método `fetch()` é um pseudónimo para `mysqli_fetch_array`, o que significa que os dados obtidos por este método são colocados num array

```
$result = $sqldb->query("SELECT * from State");
```

## Obtenção de dados MySQL

Podemos também usar comandos pré-preparados para prevenir injeções SQL

```
$result = $sqlldb->prepare("SELECT * from State WHERE state = ?");  
$state="Wisconsin";  
$result->bind_param("s", $state);  
$result ->execute();
```

# PHP - PDO

## Obtenção de dados

Para visualizar os dados podemos iterar o array:

```
while ($row = $result->fetch())  
{  
    print_r($row);  
}
```

# PHP - PDO

## Obtenção de dados

Os comandos podem retornar resultados de diferentes modos:

PDO::FETCH\_ASSOC: retorna um array indexado por nome da coluna

PDO::FETCH\_NUM: retorna um array indexado por número de coluna

PDO::FETCH\_BOTH (default): retorna um array indexado por nome e número de coluna

Alterar o modo por omissão tem que ser feito de cada vez que a ligação é efetuada:

```
$sqlldb->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,  
PDO::FETCH_ASSOC);
```

## Transações

Nem todas as bases de dados suportam transações, portanto o PDO necessita de correr no modo “auto-commit” aquando da primeira ligação

Se necessitarmos de utilizar transações, devemos usar o método **beginTransaction()** para iniciar uma

```
$sqldb->beginTransaction();
```

// os comandos são aqui

```
$sqldb->commit; // ou $sqldb->rollBack();
```



## Gestão de erros

O objeto PDO oferece 3 estratégias diferentes para lidar com erros:

**PDO::ERRMODE\_SILENT** O modo por omissão, nenhum erro é mostrado. Podemos usar `errorCode()` e `errorInfo()` para posteriormente inspecionar o erro

**PDO::ERRMODE\_WARNING** Similar ao anterior, mas mostra avisos

**PDO::ERRMODE\_EXCEPTION** Adicionalmente a indicar o código do erro, o PDO lança uma exceção `PDOException` e define as suas propriedades para refletir o erro e a sua informação

# PHP - PDO

## Gestão de erros

Definir a estratégia de gestão de erros:

```
$sqlldb->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Utilizando exceções PDO :

```
try {  
    $result = $sqlldb->prepare("SELECT * FROM mytable");  
    $result->execute(array($name));  
    $result = $result->fetchAll()  
} catch (PDOException $e) {  
    // Fazer alguma coisa...  
    echo $e->getMessage(); }
```

# PHP – FICHEIROS & I/O

Utilizar ficheiros é uma operação muito útil no dia-a-dia

Existem 4 funções principais relacionadas com ficheiros:

- Abrir um ficheiro

- Ler um ficheiro

- Escrever um ficheiro

- Fechar um ficheiro

# PHP – FICHEIROS & I/O

A função **fopen()** é utilizada para abrir um ficheiro

Requer 2 argumentos, 1 para indicar o ficheiro e outro para indicar o modo

Modo	Propósito
r	Opens the file for reading only [...]
r+	Opens the file for reading and writing [...]
w	Opens the file for writing only [...]
w+	Opens the file for reading and writing only [...]
a	Similar to w, but places the pointer at the end of the file
a+	Similar to w+, but places the pointer at the end of the file

## PHP – FICHEIROS & I/O

Para ler um ficheiro, podemos chamar a função **fread()**

Esta função requer dois argumentos, que são o ponteiro do ficheiro e o tamanho do ficheiro a ler, em bytes

O tamanho do ficheiro pode ser calculado com a função **filesize()** que toma o nome do ficheiro como argumento e devolve o tamanho expresso em bytes

## PHP – FICHEIROS & I/O

Passos necessários para ler um ficheiro em PHP:

Abrir o ficheiro utilizando a função **fopen()**

Obter o tamanho de um ficheiro utilizando a função **filesize()**

Ler o ficheiro utilizando a função **fread()**

Fechar o ficheiro utilizando a função **fclose()**

# PHP – FICHEIROS & I/O

## Exemplo:

```
<?php
$filename = "C:\\users\\guest\\mp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
    echo ( "Error in opening file" );
    exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );
fclose( $file );
echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>
```

## PHP – FICHEIROS & I/O

Podemos escrever um novo ficheiro ou adicionar texto a um existente com a função **fwrite()**

Esta função requer 2 argumentos, a especificação de um ponteiro para o ficheiro e o texto a ser escrito

Opcionalmente um terceiro argumento pode ser incluído para especificar o comprimento dos dados a escrever



# PHP – FICHEIROS & I/O

Exemplo:

```
<?php
$filename = "C:\\users\\guest\\mp.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
    echo ( "Error in opening new file" );
    exit();
}
fwrite( $file, "This is a simple test\n" );
fclose( $file );
?>
```

# PHP – FICHEIROS & I/O

Exemplo:

```
<?php
if( file_exist( $filename ) )
{
    $filesize = filesize( $filename );
    $msg = "File created with name $filename ";
    $msg .= "containing $filesize bytes";
    echo ( $msg );
}
else
{
    echo ("File $filename does not exist" );
}
```