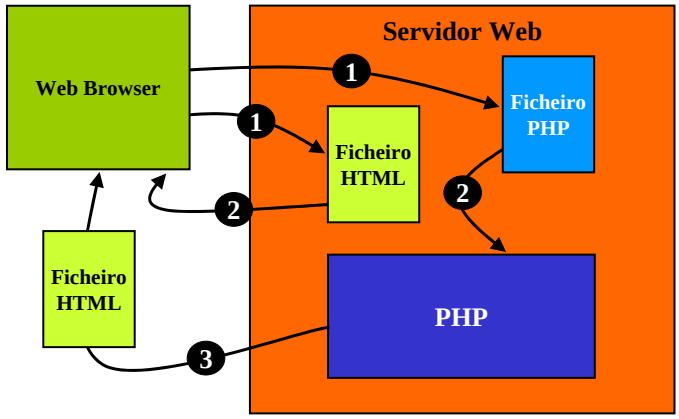
Bases de Dados

Parte VII
Interface PHP ao MySQL

O Que é o PHP?

- O PHP é uma linguagem de programação especialmente desenhada para servidores *web* (***server-side scripting language***). O PHP funciona como que uma extensão aos servidores *web* a qual permite processar informação antes de enviar o resultado para o lado dos clientes. O código PHP é embebido directamente nas páginas HTML, sendo interpretado pelo servidor *web* sempre que estas são requisitadas.
- O PHP foi concebido em 1994 por Rasmus Lerdorf. Originalmente, o acrónimo PHP significava *Personal Home Page* tendo sido alterado mais tarde para *PHP Hypertext Preprocessor* (definição recursiva ao estilo GNU).
- O PHP é *open source*, ou seja, temos acesso ao código, podemos utilizá-lo, modificá-lo e redistribui-lo.
- Uma das mais valias do PHP é a simplicidade com que é possível escrever aplicações *web* com suporte de base de dados. O PHP inclui suporte para um vasto conjunto de bases de dados: MySQL, PostgreSQL, mSQL, Oracle, Sybase, dBase, FilePro, Informix, InterBase, ODBC, ...

Como Funciona o PHP?



Como Funciona o PHP?

- Quando um servidor *web* recebe um pedido HTTP como

  ```
  GET teste.html HTTP/1.0
  ```

  o conteúdo do ficheiro **`teste.html`** é devolvido ao requisitante.
- Quando o pedido HTTP é para um ficheiro PHP, como por exemplo,

  ```
  GET teste.php HTTP/1.0
  ```

  o servidor interpreta o conteúdo do ficheiro **`teste.php`** e o resultado é que é enviado ao requisitante. Por exemplo, se **`teste.php`** contiver:

  ```
  <html><body>
  <?php echo '<b>Hello world!</b>'; ?>
  </body></html>
  ```

  o código entre **`<?php`** e **`?>`** é interpretado e o HTML resultante é:

  ```
  <html><body>
  <b>Hello world!</b>
  </body></html>
  ```

Variáveis e Operadores em PHP

- Em PHP, as variáveis não são declaradas, não têm tipo (são polimórficas) e são representadas pelo símbolo **$** antes do seu nome.

```
$a = 123;        // número inteiro
$a = -123;       // número negativo
$a = 1.23;       // número em vírgula flutuante
$b = TRUE;       // valor lógico
$c = 'hello';    // string 'hello'
$c = "world";    // string 'world'
$d = 'a = $a';   // string 'a = $a'
$d = "a = $a";   // string 'a = 1.23' ($a é substituído)
```

- Operadores aritméticos: {**+**, **–**, *****, **/**, **%**}
- Operador de concatenação de strings: {**.**}
- Operadores lógicos: {**and**, **&&**, **or**, **||**, **!**}
- Operadores de comparação: {**==**, **!=**, **<**, **<=**, **>=**, **>**}

Vectores em PHP

- Vectores ordenados

```
$a = array();        // cria um vector vazio
$a = array('um', 2); // cria um vector e inicia-o
echo $a[0];          // devolve 'um'
echo $a[1];          // devolve 2
$a[2] = 'três';      // cria novo elemento
$a[] = 'quatro';     // cria novo elemento no fim do vector
```

- Vectores associativos

```
$b = array('Ana' => '12-05-1978', 'Pedro' => '02-02-1984');
echo $b['Ana'];      // devolve '12-05-1978'
echo $b['Pedro'];    // devolve '02-02-1984'
$b['Rui'] = '30-12-1980';           // cria novo elemento
echo 'A Ana nasceu em ' . $b['Ana']; // concatenar strings
```

Estruturas de Controle em PHP

- Execução condicional

```
if ($nome == 'Rui') {
   echo '<b>Olá Mestre!</b>';
} else {
   echo "<b>Olá $nome!</b>"
}
```

- Ciclos condicionais

```
$cont = 1;
while ($cont <= 10) {
   echo "$cont ";
   $cont++;  // o mesmo que '$cont = $cont + 1'
}
```

- Ciclos numeráveis

```
for ($cont = 1; $cont <= 10; $cont++) {
   echo "$cont ";
}
```


Mudança de Modo em PHP

- É possível alternar as estruturas de controle do PHP com código HTML.

```
<?php if ($nome == 'Rui') { ?>
   <b>Olá Mestre!</b>
<?php } else { ?>
   <b>Seja bem vindo!</b>
<?php } ?>
```

- Para o *if-then-else* existe uma sintaxe alternativa que permite tornar o código mais legível.

```
<?php if ($nome == 'Rui'): ?>
   <b>Olá Mestre!</b>
<?php else: ?>
   <b>Seja bem vindo!</b>
<?php endif; ?>
```

Passar Informação em PHP

- A forma mais simples de passar informação adicional no pedido de uma página é utilizando o próprio URL. Considere, por exemplo, o seguinte *link* estático:

```
<a href="ola.php?nome=Ana">Ana</a>
```

- Para além da referencia à página **ola.php**, o *link* acima instancia a variável **nome** com o valor **Ana** (texto após o sinal **?**). As variáveis instanciadas desta forma são guardadas pelo PHP no vector associativo **$_GET**. A leitura da variável **nome** pela página **ola.php** poderia ser feita do seguinte modo:

```
<?php
   $nome = $_GET['nome'];
   echo "<b>Olá $nome!</b>"
?>
```

- O PHP permite passar várias variáveis num URL. Para tal basta utilizar o símbolo & (**&** em HTML) para separar os pares variável/valor.

```
<a href="ola.php?nomep=Ana&amp;nomef=Sousa">Ana Sousa</a>
```


Passar Informação em PHP

- Também podemos passar informação por utilização de formulários em HTML:

```
<form action="ola.php" method="get">
Nome: <input type="text" name="nome"><br>
<input type="submit" value="OK">
</form>
```

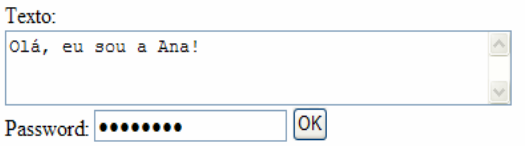
Nome: Ana
OK

- Neste caso, o método GET do formulário (**method="get"**) produz o mesmo efeito do *link* estático **"ola.php?nome=Ana"**.
- A mais valia dos formulários é que ao inserirmos diferentes nomes nas caixas de texto conseguimos gerar diferentes pedidos de páginas (*link* dinâmico).

Passar Informação em PHP

- Por vezes pode não ser viável ou desejável passar informação através do URL. Isso acontece quando o texto a enviar excede o tamanho máximo do URL ou quando esse texto diz respeito a *passwords*. Nesses casos, a alternativa é usar o método POST dos formulários (**`method="post"`**):

```
<form action="insere.php" method="post">
Texto:<br>
<textarea name="texto" rows="3" cols="50"></textarea><br>
Password: <input type="password" name="pwd">
<input type="submit" value="OK">
</form>
```



Passar Informação em PHP

- No caso anterior, o método POST do formulário produz o mesmo efeito do *link* estático **`"insere.php"`**. A desvantagem do método POST é que inviabiliza a possibilidade de fazer *bookmark* das páginas criadas a partir do formulário.
- As variáveis instanciadas pelo método POST são guardadas pelo PHP no vector associativo **`$_POST`**. A leitura das variáveis **`texto`** e **`pwd`** pela página **`insere.php`** poderia ser feita do seguinte modo:

```
<?php
   $texto = $_POST['texto'];
   $pwd = $_POST['pwd'];
   insere_bd($texto, $pwd);
?>
```

Passar Informação em PHP

- O PHP tem ainda um vector associativo **$_REQUEST** que guarda tanto as variáveis instanciadas pelo método GET como pelo método POST. Este vector permite que o código PHP fique independente do método utilizado no formulário.

```
<?php
   $texto = $_REQUEST['texto'];
   $pwd = $_REQUEST['pwd'];
   insere_bd($texto, $pwd);
?>
```


Passar Informação em PHP

- O PHP permite ainda que as variáveis instanciadas pelos métodos GET e POST possam ser criadas de modo automático evitando assim o acesso aos vectores **$_GET**, **$_POST** ou **$_REQUEST**.

```
<?php
   insere_bd($texto, $pwd); // vars criadas automaticamente
?>
```

- Para que isso aconteça é necessário activar a opção

```
register_globals = On
```

  no ficheiro de configuração do PHP (**php.ini**).

- A função **phpinfo()** permite visualizar todas as opções e valores por defeito do PHP para o servidor *web* em execução.

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

Passar Informação em PHP

- Para passar informação entre páginas podemos mostrar ou esconder valores em formulários.

```
<?php ler_valores_bd($id, $nome); ?>
<form action="altera.php" method="post">
Nome:<input type="text" name="nome"
     value="<?php echo htmlspecialchars($nome); ?>"><br>
<input type="hidden" name="id" value="<?php echo $id; ?>">
<input type="submit" value="OK">
</form>
```

- A função **`htmlspecialchars()`** é necessária porque permite converter caracteres especiais do HTML (como **`<`**, **`>`** ou **`"`**) nos seus códigos respectivos. Por exemplo, **`htmlspecialchars('<br>')`** devolve **`'<br>'`**.

Passar Informação em PHP

- Outra forma de passar informação entre páginas é adicionar essa informação aos *links* da página corrente. Por exemplo, para passar a variável **`nome`** para a página **`ola.php`** podemos fazer o seguinte:

```
<?php $nome = $_GET['nome']; ?>
<a href="ola.php?nome=<?php echo urlencode($nome); ?>">
   Carregue para continuar!</a>
```

- A função **`urlencode()`** é necessária porque permite converter caracteres especiais (por exemplo espaços) para caracteres válidos do URL. Por exemplo, se a variável **`nome`** tiver o valor **`'Ana Sousa'`**, a função **`urlencode($nome)`** devolve **`'Ana+Sousa'`**.

Passar Informação em PHP

- Podemos também condicionar o acesso a uma página quando não é fornecida toda a informação. Por exemplo, para garantir que o acesso a uma página traz a informação relativa ao nome do utilizador podemos fazer o seguinte:

```
<?php if (isset($_GET['nome'])): ?>  // URL com ?nome=
   <?php $nome = $_GET['nome']; ?>
   <a href="ola.php?nome=<?php echo urlencode($nome); ?>">
      Carregue para continuar!</a>
<?php else: ?>  // URL sem ?nome=
   <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
      method="get"> // PHP_SELP para carregar a mesma página
   Nome: <input type="text" name="nome"><br>
   <input type="submit" value="OK">
   </form>
<?php endif; ?>
```


Estruturar o Código em PHP

- Uma forma simples de estruturar o código é utilizar a funcionalidade do PHP que permite incluir um ficheiro noutro (***include files***). A ideia de utilizar *include files* é evitar escrever repetidamente os mesmos blocos de código.

```
<?php include 'ficheiro.inc.php'; ?>
<?php require 'ficheiro.inc.php'; ?>
<?php include_once 'ficheiro.inc.php'; ?>
<?php require_once 'ficheiro.inc.php'; ?>
```

- Quando o PHP encontra uma expressão **`include`** ou **`require`** ele interpreta o ficheiro correspondente tal como se este fizesse parte do ficheiro principal. Se o ficheiro não existir ou não tiver permissões de leitura, no caso do **`include`** o PHP dá um aviso e continua a execução, no caso do **`require`** o PHP dá um erro e interrompe a execução.
- Quando o PHP encontra uma expressão **`include_once`** ou **`require_once`** o ficheiro correspondente apenas é interpretado se este não tiver já sido incluído para o pedido de página em processamento. É útil para tarefas que apenas devem ser realizadas uma única vez, como por exemplo, estabelecer a ligação com a base de dados.

Estruturar o Código em PHP

- Ficheiros incluídos têm acesso a todas as variáveis do ficheiro principal e vice-versa. Apesar de isto parecer uma vantagem, deve evitar-se partilhar variáveis entre ficheiros incluídos, pois para além de dificultar a compreensão do código, pode levar a situações de sobreposição indesejada de variáveis.

```
<html><body>
<?php
   $x = 3; $y = 5;
   include 'area.inc.php';
   echo $area;
?>
</body></html>
```

Ficheiro **`'area.inc.php'`**

```
<?php $area = $x * $y; ?>
```

- Uma boa estratégia para evitar sobreposições é atribuir nomes pouco comuns às variáveis dos ficheiros incluídos, como por exemplo, nomes do tipo **`$_var`**.

Estruturar o Código em PHP

- Outra forma simples de estruturar o código é definir **funções**. Em PHP, as funções podem ser definidas em qualquer ponto do ficheiro, incluindo depois do código que as utiliza. A prática comum é definir as funções no topo do ficheiro.

```
<?php
   function area($comp, $alt) {
      return $comp * $alt;
   }
?>
<html><body>
<?php
   $x = 3; $y = 5;
   $area = area($x, $y);
   echo $area;
?>
</body></html>
```

Estruturar o Código em PHP

- Para evitar escrever repetidamente as mesmas funções, podemos definir as funções em ficheiros incluídos. A definição de funções em ficheiros incluídos é a melhor forma de evitar potenciais situações de sobreposição indesejada de variáveis.

```
<?php include 'area.inc.php'; ?>
<html><body>
<?php
   $x = 3; $y = 5;
   $area = area($x, $y);
   echo $area;
?>
</body></html>
```

Ficheiro `'area.inc.php'`

```
<?php
   function area($comp, $alt) { return $comp * $alt; }
?>
```


Estruturar o Código em PHP

- Por defeito, todas as variáveis duma função são locais à função. No entanto, por vezes, pode ser útil ou necessário uma função ter acesso às variáveis globais do ficheiro principal.

```
<?php
   function novo_texto($texto) {
      global $pwd;  // declara que $pwd é uma variável global
      insere_bd($texto, $pwd);
   }
?>
<html><body>
<?php
   $pwd = $_REQUEST['pwd'];
   novo_texto('Hello world!');
   novo_texto('Seja bem vindo!');
?>
</body></html>
```

Estruturar o Código em PHP

- Quando se utilizam variáveis para guardar valores constantes é preferível usar explicitamente **constantes** em lugar de variáveis. Em PHP, as constantes têm visibilidade global e devem ser definidas por nomes em maiúsculas. A prática comum é definir as constantes no topo do ficheiro ou em ficheiros incluídos.

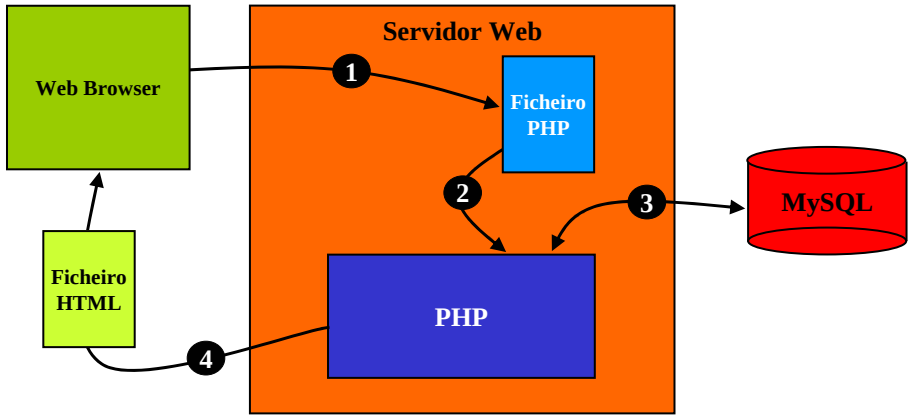
```
<?php
   define('MAX_ALT', 100);
?>
<html><body>
<?php
   $x = 3; $y = MAX_ALT;
   $area = area($x, $y);
   echo $area;
?>
</body></html>
```


Controle de Erros em PHP

- Existem funções do sistema que podem não suceder quando são executadas, como por exemplo, a função para estabelecer a ligação com uma determinada base de dados. Quando isso acontece, normalmente a função apresenta uma mensagem de erro, mas o PHP continua a executar o restante código.
- Para evitar que as funções do sistema apresentem as suas mensagens de erro podemos utilizar o sinal @ antes da chamada à função.
- Para interromper a execução do PHP e apresentar mensagens próprias de erro podemos utilizar as funções **`exit()`** ou **`die()`**.

```
if (!mysql_connect(...))   // apresenta erro se falhar
   exit();                 // interrompe a execução
...
if (!@mysql_connect(...)) // não apresenta erro se falhar
   exit('<b>ERRO:</b> Falhou ao ligar à base de dados!');
```

Acesso ao MySQL em PHP



API PHP do MySQL

- Estabelecer uma ligação com o servidor de MySQL.

```
$nome_ligação = mysql_connect($host, $user, $password);
```

- Seleccionar a base de dados a utilizar.

```
mysql_select_db($nome_bd, $nome_ligação);
```

- Executar um comando SQL e obter o seu resultado. Conjunto de comandos são processados como uma transacção.

```
$nome_resultado = mysql_query($comando_sql, $nome_ligação);
```

- Libertar um resultado. Por defeito, todos os resultados são libertados quando o PHP termina de processar um ficheiro.

```
mysql_free_result($nome_resultado);
```

- Fechar uma ligação. Por defeito, todas as ligações são fechadas quando o PHP termina de processar um ficheiro.

```
mysql_close($nome_ligação);
```

API PHP do MySQL

- Obter o número de tuplos de um resultado.

```
$num = mysql_num_rows($nome_resultado);
```

- Obter o número de atributos de um resultado.

```
$num = mysql_num_fields($nome_resultado);
```

- Obter o número de tuplos modificados durante a execução do último comando do tipo INSERT, DELETE ou UPDATE.

```
$num = mysql_affected_rows();
```


API PHP do MySQL

- Código-tipo de acesso ao MySQL com PHP.

```
<?php
   $lig = mysql_connect('localhost', 'guest', 'mypwd');
   if (!$lig) exit('Falhou ao ligar à base de dados!');
   if (!@mysql_select_db('empresa', $lig))
      exit('Falhou ao seleccionar a base de dados!');
   $query = 'SELECT * FROM EMPREGADO';
   $res = mysql_query($query, $lig);
   if (!$res) exit('Falhou ao executar consulta!');
   echo 'Existem '.mysql_num_rows($res).' empregados!';
   mysql_free_result($res);
   mysql_close($lig);
?>
```

API PHP do MySQL

- Processamento de transacções.

```
<?php
   $lig = mysql_connect('localhost', 'guest', 'mypwd');
   if (!$lig) exit('Falhou ao ligar à base de dados!');
   if (!@mysql_select_db('empresa', $lig))
      exit('Falhou ao seleccionar a base de dados!');
   $query = "UPDATE EMPREGADO SET Salário = 2000
             WHERE NumBI = '235635462';";
   $query .= "UPDATE EMPREGADO SET Salário = 3000
             WHERE NumBI = '234599982';";
   $res = mysql_query($query, $lig);
   if (!$res) exit('Falhou ao executar transacção!');
   echo 'Foram alterados '.mysql_affected_rows().' tuplos!';
?>
```


API PHP do MySQL

- Obter o próximo tuplo de um resultado como um vector ordenado.

```
$tuplo = mysql_fetch_row($nome_resultado);
```

- Obter o próximo tuplo de um resultado como um vector associativo.

```
$tuplo = mysql_fetch_assoc($nome_resultado);
```

- Obter o próximo tuplo de um resultado simultaneamente como um vector ordenado e associativo.

```
$tuplo = mysql_fetch_array($nome_resultado);
```

API PHP do MySQL

- Processamento de tuplos.

```
<?php
   $query = 'SELECT NumBI, Salário FROM EMPREGADO
             WHERE NumDep = 4';
   $res = mysql_query($query, $lig);
   $tup = mysql_fetch_row($res);
   echo 'linha 0 coluna 0: '.$tup[0];
   echo 'linha 0 coluna 1: '.$tup[1];
   $tup = mysql_fetch_assoc($res);
   echo 'linha 1 coluna 0: '.$tup['NumBI'];
   echo 'linha 1 coluna 1: '.$tup['Salário'];
   $tup = mysql_fetch_array($res);
   echo 'linha 2 coluna 0: '.$tup[0];
   echo 'linha 2 coluna 1: '.$tup['Salário'];
?>
```


API PHP do MySQL

- Processamento de tuplos.

```
<?php
   $query = 'SELECT NumBI, Salário FROM EMPREGADO
             WHERE NumDep = 4';
   $res = mysql_query($query, $lig);
   echo "<table>\n";                  // \n muda de linha
   while ($tup = mysql_fetch_row($res)) {
      echo "\t<tr>\n";                // \t faz tabelação
      foreach ($tup as $val)          // em cada iteração, $val...
         echo "\t\t<td>$val</td>\n";  // ...é um elemento de $tup
      echo "\t</tr>\n";
   }
   echo "</table>\n";
?>
```

Sessões em PHP

- O protocolo HTTP não mantém o estado entre pedidos de páginas. Quando um utilizador pede uma página e depois outra, o protocolo HTTP não nos fornece nenhum mecanismo para sabermos que ambos os pedidos são do mesmo utilizador.
- O conceito de sessão em PHP é definir um período de tempo para o qual é possível associar informação aos pedidos de um mesmo utilizador.
- Ao iniciar-se uma nova sessão com um utilizador, o PHP cria um identificador único (sequência de letras e números) que é enviado de forma automática em todos os pedidos seguintes desse utilizador, permitindo assim a sua correcta identificação.
- Para cada nova sessão, o PHP cria um vector associativo **`$_SESSION`** no qual é possível guardar informação da sessão. Os valores de uma determinada sessão guardados em **`$_SESSION`** são apenas visíveis para essa sessão.

```php
<?php
   $_SESSION['pwd'] = 'mypwd';
?>
```


Sessões em PHP

- A função **`session_start()`** recupera a sessão associada ao identificador recebido (repõe o vector **`$_SESSION`**) ou, caso este não exista, inicia uma nova sessão. Esta função deve ser executada antes de qualquer conteúdo HTML.
- A função **`session_destroy()`** termina a sessão corrente e remove toda a informação a ela associada. Não destrói o identificador da sessão nem o estado do vector **`$_SESSION`**.
- A função **`unset($var)`** destrói a variável **`$var`**. Pode ser usado para remover variáveis de sessão.

```php
session_start();      // recupera ou inicia uma nova sessão
$_SESSION['pwd'] = 'mypwd'; // cria variável de sessão
$_SESSION['login'] = TRUE; // cria variável de sessão
unset($_SESSION['pwd']);   // remove variável de sessão
session_destroy();         // termina a sessão corrente
echo $_SESSION['login'];   // devolve 'TRUE'
$_SESSION = array();       // apaga o vector $_SESSION
```

Sessões em PHP

```
<?php session_start(); ?>
<html><body>
<?php
   if (isset($_POST['pwd']))
      if($_POST['pwd'] == 'mypwd')
         $_SESSION['login'] = TRUE;
   if (isset($_GET['logout']))
      unset($_SESSION['login']);
   if (isset($_SESSION['login'])) { ?>
      ...   // a informação apresentada aqui é segura
      <a href="<?php echo $_SERVER['PHP_SELF']; ?>?logout=1">Logout</a>
<?php } else { ?>
   <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
   Password: <input type="password" name="pwd"><br>
   <input type="submit" value="OK">
   </form>
<?php } ?>
</html></body>
```