

Programação II

Formatação, Validação e Sanitização de Dados
Segurança
Uploads



Formatação de dados

- **strtoupper()**
 - Transforma uma string em maiúsculas.
- **strtolower()**
 - Transforma uma string em minúsculas.
- **ucfirst()**
 - Transforma somente a primeira letra para maiúscula.
- **ucwords()**
 - Transforma a primeira letra de cada palavra em maiúsculas

Formatação de dados

- **n12br()**

- Retorna uma string com '
' inserido antes de todas as newlines (\n). Útil quando exibimos dados vindos de campos do tipo <textarea>, onde o usuário pode quebrar linhas usando ENTER. Sem este comando as quebras são ignoradas, pois o navegador não interpreta \n como quebra de linha no momento de renderizar a página.

```
/* suponha que o usuário digitou no campo observações:
```

```
    Devolver o livro
```

```
    de engenharia.
```

```
    Pegar outro.
```

```
*/
```

```
$obs = nl2br($_POST['observacoes']);
```

```
// Devolver o livro<br />de engenharia.<br /><br />Pegar  
outro.
```

Validação de dados

- **strlen()**

- Retorna o número de caracteres de uma string.

```
if (strlen($_POST["login"]) < 5) {  
    $erro = "O login deve ter pelo menos 5 caracteres!";  
} else {  
    $login = $_POST["login"];  
}
```

- **strstr()**

- Acha a primeira ocorrência de uma string. Se não encontrá-la, retorna falso.

```
if (!strstr($_POST["email"], "@")) {  
    $erro = "O e-mail deve possuir o caractere @!";  
} else {  
    $email = $_POST["email"];  
}
```

Validação de dados

- **empty()**

- Retorna verdadeiro se o conteúdo da variável está vazio.

```
if (empty($_POST["name"])) {  
    $erro = "O campo nome é obrigatório!";  
} else {  
    $name = $_POST["name"];  
}
```

- **explode()**

- Divide um dado em múltiplas porções, baseado num caractere separador. As partes são armazenadas em um array.

```
$data = "20/11/2017";  
$data2 = explode("/", $data);  
// array(0=>20, 1=>11, 2=>2017)
```

Validação de dados

- **checkdate()**

- Valida uma data baseada nos valores informados para o mês, dia e ano (nesta ordem).

```
$data = explode('/', '14/06/2018');  
if(checkdate($data[1], $data[0], $data[2])){  
    echo "Data válida";  
}  
else{  
    echo "Esta data não existe";  
}
```

- O mês deve estar entre 1 e 12; o dia deve estar dentro do número permitido de dias do mês em questão, levando em conta o ano bissexto; o ano deve estar entre 1 e 32767.

Sanitização (limpeza) de dados

- **trim()**

- Remove espaços desnecessários (em branco, tabulações, nova linha) à esquerda e à direita.

```
$nome = "  Fulano  ";  
$nome = trim($nome); // "Fulano"
```

- **str_replace()**

- Substitui todas as ocorrências em uma string por outra string.

```
$data = "20.11-2017";  
$data2 = str_replace(array(".", "-"), "/", $data); // 20/11/2017
```

- **htmlentities()**

- Converte todos os caracteres especiais em suas representações de entidade HTML equivalentes.

```
$entrada = "A tag <a> é usada para âncoras";  
$entrada2 = htmlentities($entrada);  
// "A tag &lt;a&gt; &eacute; usada para &acirc;ncoras"
```

Sanitização (limpeza) de dados

- **htmlspecialchars()**

- Semelhante à `htmlspecialchars()`, porém, converte somente caracteres utilizados na programação em entidades HTML (aspas, &, <, >). Usado em muitos casos para tornar a string “segura”, evitando que algum usuário tente entrar com tags em campos de texto aberto (ataques conhecidos como XSS, ou *cross site scripting*).

```
$entrada = "A tag <a> é usada para âncoras";  
$entrada2 = htmlspecialchars($entrada);  
// "A tag &lt;a&gt; é usada para âncoras"
```

- Assim, se o usuário submeter o seguinte conteúdo em um campo de texto aberto:

```
<script>location.href('http://www.hacked.com')</script>
```

- o script não será utilizado, pois será transformado em:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```


Sanitização (limpeza) de dados

- **addslashes()**

- Adiciona barras invertidas antes dos caracteres aspas simples ('), aspas duplas ("), barra invertida (\) e NUL (o byte NULL). Por exemplo, para inserir o nome O'reilly em um banco de dados, será necessário escapá-lo com addslashes, caso contrário, o apóstrofo será interpretado como aspas simples e provocará um erro. O dado ficará como O\'reilly (isto é apenas para colocar os dados no banco de dados, a \ não será inserida).

```
$str = addslashes("O'reilly");  
echo $str;  
// imprime O\'reilly
```

- Se a diretiva do PHP `magic_quotes_gpc` estiver configurada como `on`, `addslashes()` é executada automaticamente para todos dados de GET, POST e COOKIE.

- **stripslashes()**

- Retira barras invertidas das strings.

```
$str = stripslashes("O\'reilly");  
echo $str;  
// imprime O'reilly
```

Validação / Limpeza com filter_var

- **filter_var()**
 - Permite filtrar e validar informações. Dois filtros são possíveis:
 - *validate* (validação): têm como objetivo verificar se a informação "casa" com um padrão específico. As constantes que identificam os padrões podem ser encontradas em <http://php.net/manual/en/filter.filters.validate.php>.

```
$email = "asebben@uffs.edu.br";  
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {  
    $emailErr = "Formato de e-mail inválido";  
}
```

- *sanitize* (limpeza): retiram caracteres não permitidos no padrão que está sendo utilizado. As constantes que identificam os padrões podem ser encontradas em <http://php.net/manual/en/filter.filters.sanitize.php>.

```
echo filter_var("<b>oi</b>", FILTER_SANITIZE_STRING);  
//imprime oi, sem estar em negrito (limpa as tags)
```

Expressões Regulares

- Uma expressão regular (ou regex) é um padrão para descrever um texto (ou parte dele). Muito utilizada em diversas linguagens para identificar e validar formatos de e-mail, telefone, CEP, etc.
- O programa tenta "casar" uma string com uma expressão regular, a fim de determinar se a string se enquadra naquele formato.
- Para checar a conformidade de um valor com a expressão regular em PHP, pode-se usar `preg_match` ou, para substituições, `preg_replace`.
- O exemplo abaixo verifica se `$login` possui entre 4 e 20 caracteres e se estes são apenas letras (*case insensitive*), números e *underline*:

```
$regex = "/^[a-z\d_]{4,20}$/i";  
if(!preg_match($regex, $login)){  
    echo 'Login inválido!.';  
}
```

Expressões Regulares

- Uma regex sempre será delimitada por barras no início e fim:

```
$regex = "/php/";  
// Coincide: "php", "texto em php"  
// Não Coincide: "PHP", "Texto em ph"
```

- Para que a comparação seja case insensitive, acrescenta-se um i após a última barra:

```
$regex = "/php/i";  
// Coincide: "php", "Texto pHp", "PHP"  
// Não coincide: "Texto em ph"
```

- Múltiplas opções de caracteres podem ser especificadas entre colchetes:

```
$regex = "/p[ho]p/";  
// Coincide: "php", "pop"  
// Não coincide: "phop", "pip", "PHP"
```

- Uma faixa de letras e números também pode ser especificada entre colchetes:

```
$regex = "/p[a-z1-3]p/";  
// Coincide: "php", "pup", "pap", "pop", "p3p"  
// Não coincide: "PHP", "p5p"
```

- O ponto (.) coincide com qualquer caractere, exceto quebra de linha:

```
$regex = "/p.p/";  
// Coincide: "php", "p&p", "p(p", "p3p", "p$p"  
// Não coincide: "PHP", "phhp"
```

Expressões Regulares

- Classes pré-definidas:
 - `\d` representa um único caractere numérico entre 0 e 9.
 - `\s` representa um espaço em branco (inclui tabulações e quebras de linha).
 - `\w` representa um caractere alfanumérico (inclui underline).

```
$regex = "/p\d{2}p/";  
// Coincide: "p3p", "p7p",  
// Não coincide: "p10p", "P7p"
```

```
$regex = "/p\w{2}p/";  
// Coincide: "p3p", "pHp", "pop"  
// Não coincide: "phhp"
```

Expressões Regulares

- Alguns caracteres especiais servem para indicar que um grupo de caracteres pode ser repetido:
 - `?` Zero ou uma vez
 - `*` Zero ou mais vezes
 - `+` Uma ou mais vezes
 - `{n,m}` Entre n e m vezes
- Âncoras para indicar início e fim (e evitar que qualquer parte da string coincida):
 - `^` Início de uma string
 - `$` Final de uma string
- Uma barra invertida serve como caractere de escape, para que possamos usar os caracteres reservados sem que sejam interpretados como tal:

```
$regex = "/p\.p/";
```

Expressões Regulares

- Exemplo de uma regex para testar um endereço de e-mail:

```
"/^[a-z\d\._-]+@([a-z\d-]+\.)+[a-z]{2,4}$/i"
```

```
// Coincide: "fulano@gmail.com",  
            "beltrano2@hotmail.com.br",  
            "ciclano_5_a@xpto.co.uk"
```

```
// Não coincide: "cic@xpto@inf.com",  
                "bel.tran.no.com"
```

Expressões Regulares

Delimitador e âncora de início de string

/^

Nome do usuário – qualquer texto constituído de letras, números, pontos, underscores e traços

[a-z\d\._-]+

@

○ @ separador

Domínio (letras, números e/ou traços). Repete para incluir subdomínios.

([a-z\d-]+\.)+

[a-z]{2,6}

br, pt, com, uk, info, museum...

\$/i

âncora e delimitador de final de string, indicador de case insensitive.

Criptografia

- Dados sensíveis, como senhas, não devem ser armazenados em seu formato original (texto simples).
- Um método de criptografia unidirecional (*hash*) deve ser aplicado.
 - Unidirecional significa que um *hash* não pode ser transformado novamente na senha (ou texto) que lhe deu origem. Deste modo, a verificação é feita por meio da comparação das duas hash (uma oriunda da base de dados, e a outra da tentativa de login).
- Existem diversas formas de aplicar hash em senhas: **md5** (*message digest algorithm*), **sha1** (*secure hash algorithm 1*), hash com **sha256** e a **password API** (esta disponível desde a versão 5.5 do PHP).
- Além disso, muitos frameworks possuem seus próprios mecanismos de criptografia.

Criptografia

```
md5('teste');
```

```
// gera: 698dc19d489c4e4db73e28a713eab07b
```

```
// (32 caracteres, sempre o mesmo)
```

```
sha1('teste');
```

```
// gera: 2e6f9b0d5885b6010f9167787445617f553a735f
```

```
// (40 caracteres, sempre o mesmo)
```

```
hash('sha256', 'teste');
```

```
// gera:
```

```
46070d4bf934fb0d4b06d9e2c46e346944e322444900a435d7  
d9a95e6d7435f5
```

```
// (64 caracteres, sempre o mesmo)
```

API Password

- Abaixo, um exemplo utilizando a API password do PHP. O hash para a mesma palavra muda a cada execução, pois emprega "salts" aleatórios.

```
$senha = password_hash('teste', PASSWORD_DEFAULT);  
// gera:  
$2y$10$yPHliHlrzAH4fiBITbHm1uABlSc/6Fm06xSYLQ9NhsdmydVOe8FDa  
  
// para testar a coincidência:  
$hash =  
'$2y$10$yPHliHlrzAH4fiBITbHm1uABlSc/6Fm06xSYLQ9NhsdmydVOe8FDa';  
if (password_verify('teste', $hash)) {  
    // senha correta  
}
```

SQL Injection

- Suponha que sua aplicação possui o seguinte código, onde login é um campo do formulário:

```
$login = $_POST['login'];
```

```
$query = "SELECT * FROM registros WHERE login = '$login'";
```

- O objetivo é listar todos os registros associados àquele login. Porém, se o usuário informar como conteúdo do campo login o texto ' OR '1=1', a consulta ficaria assim:

```
$query = "SELECT * FROM registros WHERE login = ' ' OR '1=1' ";
```

- Ou seja, o usuário identificado por este login teria acesso a todos os registros associados a outros logins.

SQL Injection

- *SQL Injection* é um ataque que consiste na inserção de uma query SQL via aplicação web, através de campos de formulários ou de valores passados por URL (método GET).
 - O usuário mal intencionado pode tentar visualizar dados restritos, bem como deletar registros ou mesmo "dropar" tabelas ou databases inteiros.
- Para se proteger do *SQL Injection*, verifique se todo parâmetro vindo do *front end* do site é tratado antes que seja concatenado na query.
- Por exemplo, nunca faça simplesmente:

```
$consulta = "DELETE FROM tabela WHERE id_tabela = " . $_POST['id'];
```

- Em vez disso, trate primeiro o `$_POST[id]`, como neste exemplo:

```
if (is_numeric($_POST['id'])) {  
    $consulta = "DELETE FROM tabela WHERE id_tabela = ".$_POST['id'];  
} else {  
    die("Dados inválidos");  
}
```

SQL Injection

- Para campos com strings é aconselhável escapar toda entrada fornecida pelo usuário, checando a existência dos caracteres:
 - " (aspas duplas) ' (aspas simples) (espaços)
 - ; (ponto e vírgula) = (sinal de igual) < (sinal de menor que)
 - > (sinal de maior que) ! (ponto de exclamação)
 - (dois hifens; indica início de comentário em alguns bancos)
 - # (sustenido ou jogo-da-velha, indica início de comentário em alguns bancos)
 - // (duas barras, indica início de comentário em alguns bancos)
- ou das palavras reservadas: SELECT, INSERT, UPDATE, DELETE, WHERE, JOIN, LEFT, INNER, NOT, IN, LIKE, TRUNCATE, DROP, CREATE, ALTER, DELIMITER
- Pode-se usar **expressões regulares** de validação e as funções **addslashes**, **htmlspecialchars** e/ou **htmlentities**.

SQL Injection

- Privilégios no BD
 - É recomendável também limitar os privilégios das contas com acesso ao banco de dados, mantendo apenas os privilégios e acessos necessários.
 - Jamais se conecte com o usuário *root* em aplicações em produção.
 - Evite também dar privilégios totais (GRANT ALL) sem necessidade, apenas para facilitar o desenvolvimento, pois isso poderá ser explorado indevidamente.

SQL Injection

- *Prepared statements*
 - Outra possibilidade é utilizar mecanismos de parametrização das consultas, as chamadas *prepared statements*, ao invés do próprio código SQL concatenado aos valores das variáveis.
 - Veja no link <http://blog.alura.com.br/como-acessar-o-banco-de-dados-com-php-7/> duas formas de fazer isto: utilizando a extensão **mysqli** e também a biblioteca **PDO** (PHP Data Objects).

```
function buscar_cliente($conexao, $id){  
    $sqlBusca = "SELECT * FROM cliente WHERE id = ? "; //? significa que ali vai um  
    parâmetro  
    $statement = $conexao->prepare($sqlBusca);  
    $statement->bind_param("i", $id);           // "i" significa um parametro do tipo inteiro  
    $statement->execute();  
    $resultado = $statement->get_result();  
    return $resultado->fetch_assoc();  
}
```

- Mais informações sobre SQL Injection no manual do PHP:
https://secure.php.net/manual/pt_BR/security.database.sql-injection.php

Upload de Arquivos

- Arquivos podem ser submetidos a um servidor através de formulários, usando o método POST.
- O atributo `enctype` do formulário deve ter o valor `multipart/form-data`, que instrui o navegador a postar múltiplas seções (uma para os dados e outra para arquivos).

```
<form action="" method="POST" enctype="multipart/form-data">
```

- O campo de entrada de arquivo (*file*) cria um botão para o usuário selecionar um arquivo local.

```
<input type="file" name="figura">
```

- O atributo `MAX_FILE_SIZE` (*uppercase*) pode ser utilizado em um campo `hidden` para especificar o tamanho máximo do arquivo (em bytes). Este atributo deve ser inserido antes do campo de entrada do arquivo.

```
<input type="hidden" name="MAX_FILE_SIZE" value="30000">
```

Upload de Arquivos

- Após o envio dos dados do formulário, informações sobre o arquivo são armazenadas na variável superglobal `$_FILES[]`. Este array possui cinco elementos:
 - Código de erro associado ao arquivo:
`$_FILES['figura']['error']`
(Lista de erros: <http://www.php.net/manual/en/features.file-upload.errors.php>)
 - Nome temporário do arquivo (no servidor):
`$_FILES['figura']['tmp_name']`
 - Nome original do arquivo (na máquina cliente):
`$_FILES['figura']['name']`
 - Tamanho do arquivo em bytes:
`$_FILES['figura']['size']`
 - Tipo MIME do arquivo:
`$_FILES['figura']['type']`

Upload de Arquivos

- A função `move_uploaded_file()` move o arquivo do diretório temporário para um local "permanente". Retorna TRUE se o arquivo foi movido, ou FALSE caso contrário.

```
bool move_uploaded_file(string $arquivo, string $destino)
```

- onde:
 - `$arquivo` é o conteúdo de `$_FILES['figura']['tmp_name'];`
 - `$destino` é o caminho completo do local para onde o arquivo será movido.

```
$uploadaddir = "uploads/";  
$uploadfile = $uploadaddir.$_FILES['figura']['name'];  
  
if(!move_uploaded_file($_FILES['figura']['tmp_name'],  
$uploadfile)){  
    echo "<p>Arquivo não foi movido</p>";  
}
```