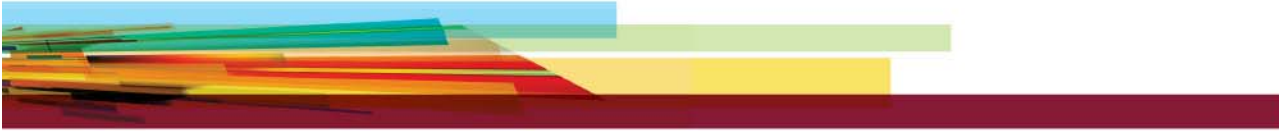




CEWP 459

PHP Programming with MySQL – Level I
Regular Expressions





Regular Expressions

eg: Regex

Online Testers

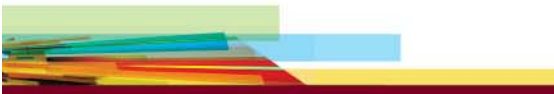
<http://regexpal.com/>

<http://regex101.com/>

<http://rubular.com/>

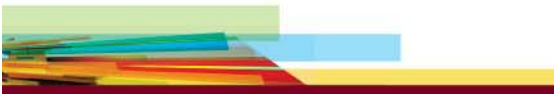
<http://www.regexr.com/>

This is really good it has a good reference library.





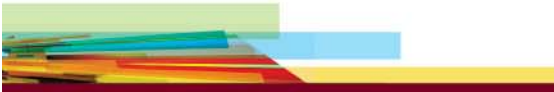
Regex in PHP

- Function: Search for matches within string
- Matching on patterns
- Two types PCRE and POSIX.
- POSIX is depreciated.
- PCRE is based on PERL regex, becoming standard.



Basics

- Literal characters are the characters you want to match or not match. 
- Metacharacters are the characters that control the logic in the regular expression. 



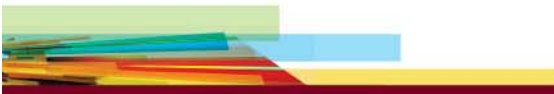
Types of searches

/ /g


Global search

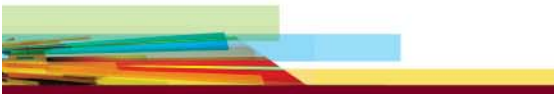
/ /

Normal search 




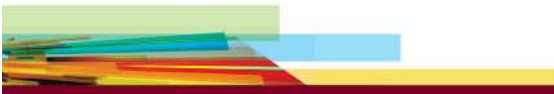
Basic Metacharacters

M	Meaning	Example
\$	Match strings that end with a pattern.	
^	Match strings that start with a pattern.	
.	Matches any character	



Metacharacters (...)

M	Meaning	Example
	Alternatives (OR expression)	
+	Match ONE or MORE	
*	Match ZERO or MORE	
()	Group expressions together, commonly used with character.	
{n,m}	Match exact number of characters	



Abbreviations

Character Class Abbreviations

`\d` Match any character in the range 0 - 9

`\D` Match any character NOT in the range 0 - 9

`\s` Match any whitespace characters (space, tab etc.).

`\S` Match any character NOT whitespace (space, tab).

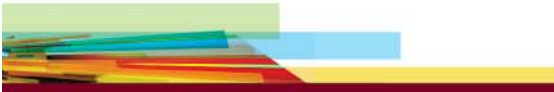
`\w` Match any character in the range 0 - 9, A - Z and a - z

`\W` Match any character NOT the range 0 - 9, A - Z and a - z

Positional Abbreviations

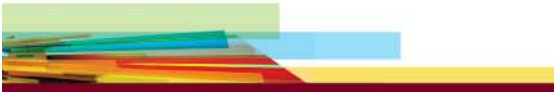
Word boundary. Match any character(s) at the beginning (`\bxx`) and/or end (`xx\b`) of a word, thus `\bton\b` will find ton but not tons, but `\bton` will find tons.

Not word boundary. Match any character(s) NOT at the beginning (`\Bxx`) and/or end (`xx\B`) of a word, thus `\Bton\B` will find wantons but not tons, but `ton\B` will find both wantons and tons.



Character Classes

- Represent a bunch of items as a single character
 - **[abc]** will match a, b, or c.
 - **[a-zA-Z0-9]** matches all characters and numbers.
- Negated class, use the ^ character
 - **[^a]** will match “hit” but not “hat”.



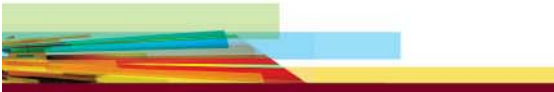
Examples

- foo
 - The string "foo"
- ^foo
 - "foo" at the start of a string
- foo\$
 - "foo" at the end of a string
- ^foo\$
 - "foo" when it is alone on a string





Examples 2

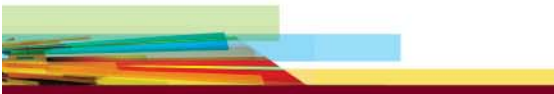
- [abc]
 - a, b, or c
- [a-z]
 - Any lowercase letter
- [^A-Z]
 - Any character that is not a uppercase letter
- (gif|jpg)
 - Matches either "gif" or "jpeg"






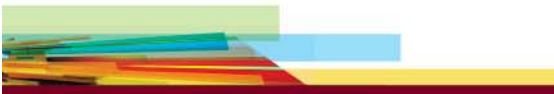
Examples 3

- `[a-z]+`
 - One or more lowercase letters
- `[0-9\.\-]` 
 - Any number, dot, or minus sign
- `^[a-zA-Z0-9_]{1,}$` 
 - Any word of at least one letter, number or _




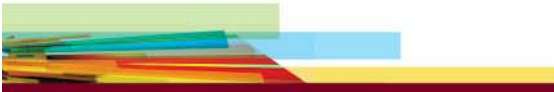
Examples 4

- `([wx])([yz])` 
 - `wy, wz, xy, or xz`
- `[^A-Za-z0-9]`
 - Any symbol (not a number or a letter)
- `([A-Z]{3}|[0-9]{4})`
 - Matches three letters or four numbers



Delimiters

- In most instances you must enclose the regular expression in delimiters.
- In PHP we're using the “/” slash character. 
- To match “text” the regex looks like this: “/text/”.

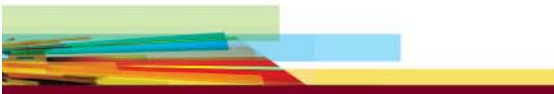


PHP (preg_match)

`preg_match (pattern, subject [, matches [, flags [, offset]]])`

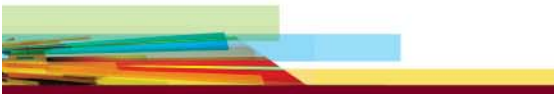
- Return value: 1 for match, 0 for non-match.

```
if (preg_match("/or/", "Hello World!",  
$matches)) {  
    echo "Match was found <br />";  
    echo $matches[0];  
}
```



PHP (preg_match)

```
if (preg_match("/ll.*/", "The History of  
Halloween", $matches)) {  
    echo "Match was found <br />";  
    echo $matches[0];  
}
```



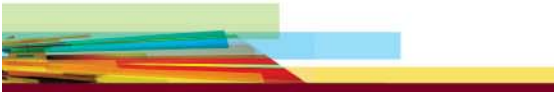
Exercise

For the following two items:



Match: jim

Not match: jam, lam, 0am

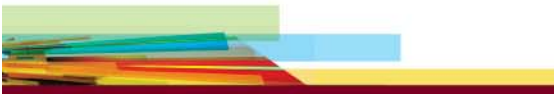


Exercise

- Match any 6 numbers.

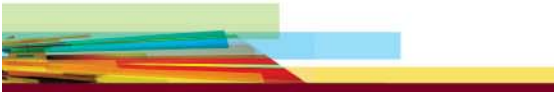
Eg: `abc000000abc`, not `abc00000abc` 

Write a program in PHP to accomplish this check. Display two lines with (MATCH or NO-MATCH) at the end of the line.



Validate Email Address

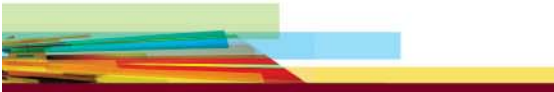
```
$email =  
"firstname.lastname@aaa.bbb.com";  
$regex = "/^[^0-9][A-z0-9_]+(.[A-z0-9_]+)*[@][A-z0-9_]+(.[A-z0-9_]+)*[.][A-z]{2,4}$/";  
if (preg_match($regex, $email)) {  
    echo "Email address is valid.";  
} else {  
    echo "Email address is not valid.";  
}
```



Regex for REPLACEMENT (preg_replace)



```
preg_replace( pattern, replacement,  
subject [, limit ])
```

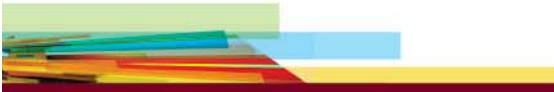
```
echo preg_replace("/([Cc]opyright)  
200(3|4|5|6)/", "$1 2007", "Copyright  
2005");
```





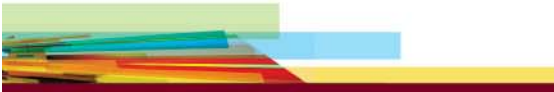
Exercise

- Write the following regex's and supply proof that they work.
 1. Validate Canadian social security number. 
 2. Validate Canadian phone number. Assume only 514 and 450. 
 3. Validate Concordia student number.
 4. Validate string to be a currency format like "\$123.55". Can be any amount.



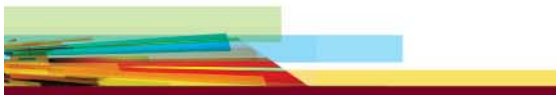
Review Regex 1

Positive	Negative
pit spot spate slap two respite	pt Pot peat part

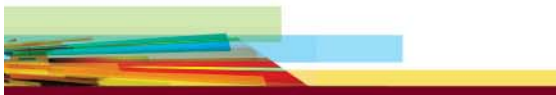




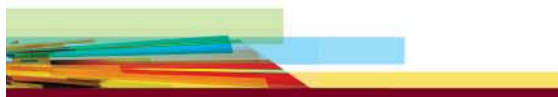
Objective	
Match	Cat.
Match	896.
Match	?=+.
Skip	abc1



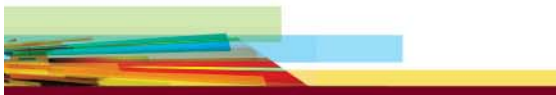
Objective	
Match	Can
Match	Man
Match	Fan
Fail	Dan
Fail	Ran
Fail	Pan



Objective	
Match	Wazzzzup
Match	Wazzzup
Fail	Wazup



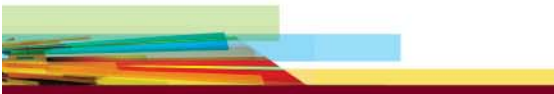
Objective	
Match	1 thing found.
Match	2 things found.
Match	3 things found.
Fail	No things found.



Regex for REPLACEMENT (preg_replace)

```
preg_replace( pattern, replacement,  
subject [, limit ])
```

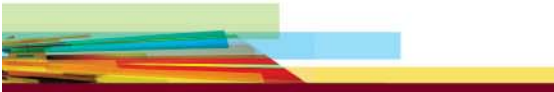
```
echo preg_replace("/([Cc]opyright)  
200(3|4|5|6)/", "$1 2007", "Copyright  
2005");
```



Regex with Callback

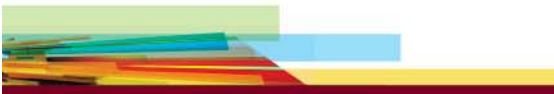


```
mixed preg_replace_callback  
(mixed $pattern ,  
callable $callback ,  
mixed $subject [, int $limit = -1 [, int &$count ]] )
```



Steps

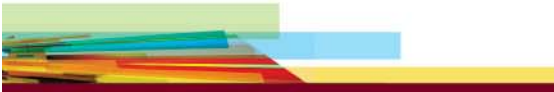
- 1) Identify your input string.
- 2) Construct a regex to find
- 3) Write a function to process the found data.
- 4) Put it all together.



Example

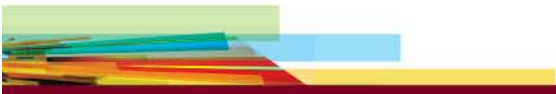


```
echo preg_replace_callback(  
    '/[A-Z]+/',  
    function ($matches) {  
        $character = reset($matches);  
        return strtolower($character);  
    },  
    'AaBbCcDd'  
);
```



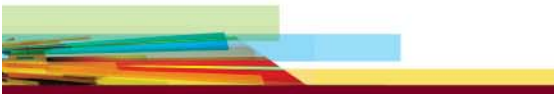
Excercise 1 (Reprise)

- Create the following REPLACEMENT statements.
 1. Replace all lowercase characters with uppercase ones.
 2. Replace all letters with spaces.
 3. Replace “(C)” or “(c)” with “Copyright”.



Exercise

- Write a program that reads all the lines of a file, and capitalizes the first letter in the first word after a period.
- Assume that there might be a space or not after the period.
- View the hints on the next slide for reminders on file manipulation.
 - Hello. this is a test. -> ello. This is a test.
 - Hello.this is a test. -> Hello.This is a test.



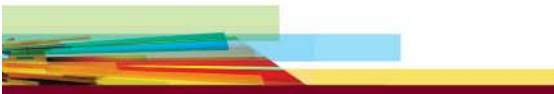
Regex Replacement with Callback

`mixed preg_replace_callback (mixed $pattern , callable $callback , mixed $subject)`

`$pattern` : Pattern to find

`$callback` : The function to call

`$subject` : The text we want to process



More on how it works;

- Once you've mastered `preg_match`, the result passed to the function is exactly the same.
- The result of `/(\d)(\d)/` on subject `345678` will first give an `ARRAY`.
 - Position 0 : The full match (34)
 - Position 1 : Parenthesis 1 (3)
 - Position 2 : Parenthesis 2 (4)

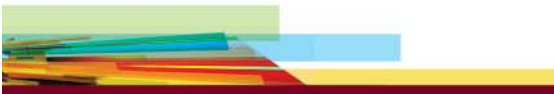


Subsequent Passes

Array ([0] => 34 [1] => 3 [2] => 4)

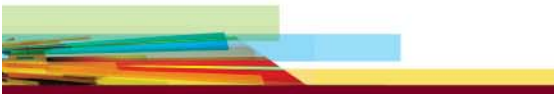
Array ([0] => 56 [1] => 5 [2] => 6)

Array ([0] => 78 [1] => 7 [2] => 8)



Exercises

- Add 1 to all numbers in a string.
 - Th1s is a string 123
 - Th2s is a string 234
- Change all years (or what looks like a year) to it's 'xx equivalent.
 - Music was a lot better from 1980 to 1989.
 - Music was a lot better from '80 to '89.





CONCORDIA.CA

