

# Regular Expressions

SET09103 Advanced Web Technologies

School of Computing  
Napier University, Edinburgh, UK  
Module Leader: Uta Priss

2008

# Outline

Basics

Wildcard and multipliers


Special characters


Negation


Other functions


Programming


## Character by character match

the cat  
  
 /the /

lathe and  
  
 /the /

lathe.  
  
 /the /

The cat  
  
 /the /

The cat  
  
 /the / i

NO match

NO match

Note: “i” at the end means “ignore case”

# Wildcard and multipliers

. stands for “any character”.

# Wildcard and multipliers

. stands for “any character”.

Multipliers:

+ stands for “at least one character”

\* stands for “any number of characters (including 0)”

? stands for “at most one character” (i.e. either none or once)

{**n,m**} stands for “at least n times, at most m times”

## Examples:

the  
↑↑↑  
/t.e/

te  
↑ ↘  
/t.\*e/

the  
↑↑↑↑  
/t.\*e/

thhhhhhhe  
↑ ↗ ↘  
/t.\*e/

the  
↑↑↑↑  
/t.+e/

thhhhhhhe  
↑ ↗ ↘  
/t.+e/

te  
↑ ↘  
/t.?e/

the  
↑↑↑↑  
/t.?e/

the      thhe  
↑ ↘ ↘    ↑ ↘ ↘  
/t.{1,2}e/ /t.{1,2}e/

# Exercise

What does `/..\19../` match:

“12.1000” or “123.1900” or “12.2000”

# Exercise

What does `/..\19../` match:

“12.1000” or “123.1900” or “12.2000”

What does `/hn*ell?o W...d/i` match:

“Hello World” or “Hello Wood” or “Hell?o World”?



## Special characters

<code>\w</code>	word character (letter, digit or <code>_</code> )
<code>[a-zA-Z]</code>	letter
<code>\W</code>	non-word character
<code>[^a-zA-Z]</code>	not a letter
<code>\d</code>	digit
<code>\s</code>	space character (blank space, tab)
<code>\b</code>	word boundary
<code>^</code>	beginning of line or string
<code>\$</code>	end of line or string

abc123

↙  
^w/

abc123

↖  
^d/

abc123

↙  
/[a-z]/

abc123

↖  
/[^a-z]

the cat

↖  
^s/

the cat

↙ ↖ ↖  
^b\w+\b/

\$a="the"

the cat

↙ ↖ ↖  
^b\$a\b/

the cat

↙ ↖ ↖  
/^.\*\$/

# Exercise

Which matches two consecutive words:

`/\b\w+\b\s+\b\w+\b/`

or

`/\w+\s+\w+/`

or

`/\b\w*\b\s*\b\w*\b/`

?

# Negation

`$word !~/a/`

means that “a” must not occur in `$word` at all.

`$word =~ /^[^a]/`

means that `$word` must have one character which is not “a”.

# Examples

the

!~ /t/

No match

the

↑  
=~ /^[^t]/

match

gst0202

=~ /^[^\w]/

No match

gst0202;

↑  
=~ /^[^\w]/

match

# Substitution

the table  $\longrightarrow$  s/t/T/  $\longrightarrow$  The table

the table  $\longrightarrow$  s/t/T/g  $\longrightarrow$  The Table

<p>the table</p>  $\longrightarrow$  s/<\/?p>//g  $\longrightarrow$  the table

<p>the table</p>  $\longrightarrow$  s/<.\*>//g  $\longrightarrow$

<p>the table</p>  $\longrightarrow$  s/<.\*?>//g  $\longrightarrow$  the table

## Remembering patterns

Brackets are used for remembering patterns. The content of the first set of brackets can be retrieved with `\1`. The second set of brackets with `\2`, and so on.

Examples:

```
s/<p>(the table)</p>/\1/
```

```
/(\.)\1/
```

```
s/(\.)(\.)/\2\1/
```

## Split and Join (Implode)

```
$oldstring = "the,cat,sat,on,the,mat";  
@array = split(/,/,$oldstring);  
print @array;  
# @array = ("the","cat","sat","on","the","mat")  
$newstring = join(" ",@array);  
# $newstring ="the cat sat on the mat"
```



# Strategies

Instead of using one complicated regular expression, it is sometimes easier to use several simpler regular expressions combined with if statements.

For example: string starts with “a” and ends with “z”:

```
if ($string =~ /^a.*z$/)
```

```
if ($string =~ /^a/ and $string =~ /z$/)
```

# More Strategies

If a string needs to be processed ...

- ▶ from left to right, one character or one word at a time  
⇒ split into array, then process array.
- ▶ from left to right, in some other regular manner  
⇒ `substr()` can be used instead of regular expression.
- ▶ by checking whether some pattern exists  
⇒ use regular expressions.

# Use of regular expressions in PHP

## Searching:

```
if (preg_match("/the /i", $line, $matches)) {  
    echo $line,"<br> matches: ",$matches[0],"<br>";}
```

## Replace:

```
$line = preg_replace("/T/", 't', $line);
```

## Split:

```
$words = preg_split("/\s+"/, $line);
```

## Implode:

```
$newstring = implode(" ", $array);
```