# Regular Expressions

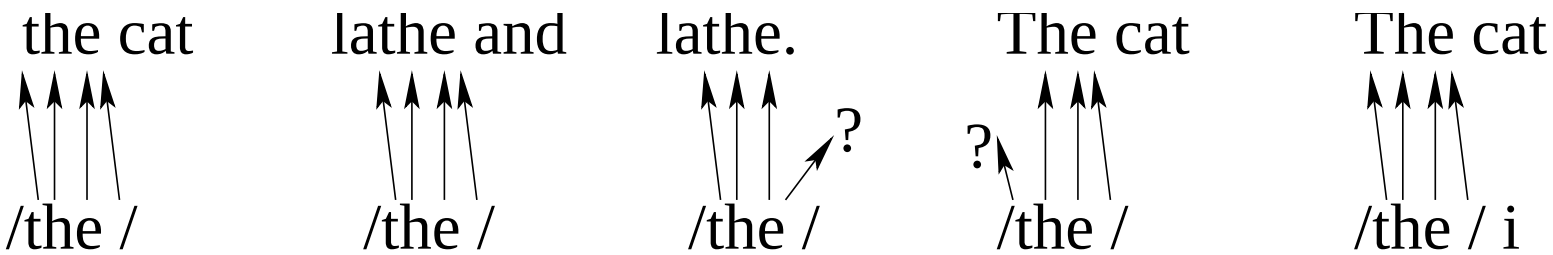
## Web Programming

Uta Priss
ZELL, Ostfalia University

2013

# Outline

# Character by character match




NO match NO match

Note: "i" at the end means "ignore case"

# Wildcard and multipliers

. stands for “any character”.

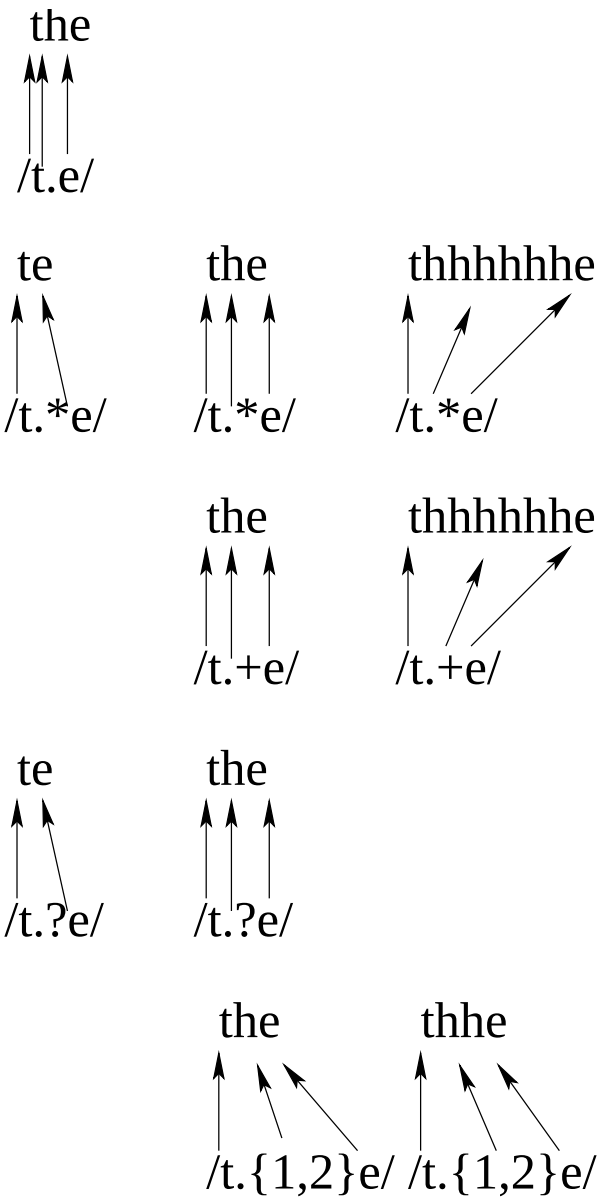
# Wildcard and multipliers

**.** 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/
thhhhhhe
/t.*e/
the
/t.+e/
thhhhhhe
/t.+e/
te
/t.?e/
the
/t.?e/
the
/t.{1,2}e/
thhe
/t.{1,2}e/

# Exercise

What does /..\.19../ match:
"12.1000" or "123.1900" or "12.2000"

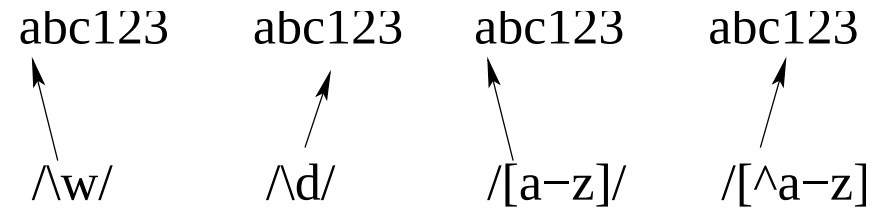
# Exercise

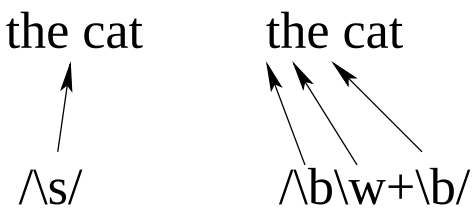
What does /..\.19../ match:
"12.1000" or "123.1900" or "12.2000"

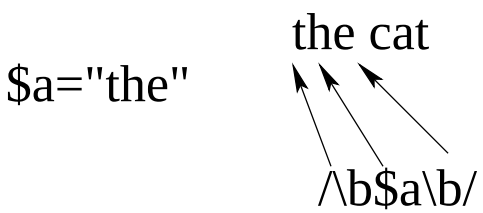
What does /hn*ell?o W...d/i match:
"Hello World" or "Hello Wood" or "Hell?o World"?

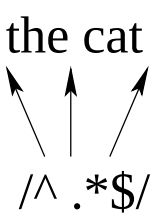
# Special characters

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


abc123
abc123
abc123
abc123
/\w/
/\d/
/[a−z]/
/[^a−z]


the cat
the cat
/\s/
/\b\w+\b/


$a="the"
the cat
/\b$a\b/


the cat
/^ .*$/

# Exercise

Which matches two consecutive words:

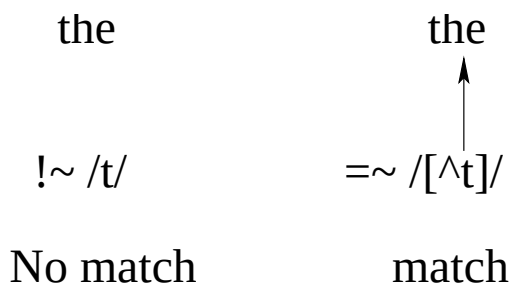
/\b\w+\b\s+\b\w+\b/
or
/\w+\s+\w+/
or
/\b\w*\b\s*\b\w*\b/
?

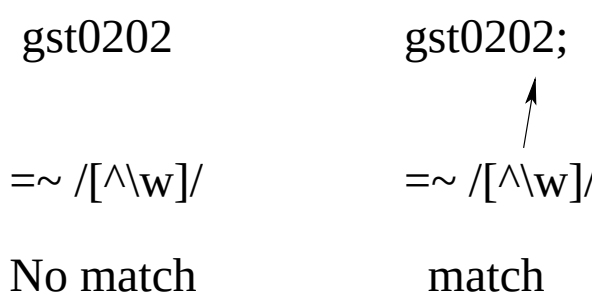
# Negation

$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



# Substitution

the table ⟶ s/t/T/ ⟶ The table

the table ⟶ s/t/T/g ⟶ The Table

the table ⟶ s/<\/?p>//g ⟶ the table

the table ⟶ s/<.*>//g ⟶

the table ⟶ s/<.*?>//g ⟶ 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
  $\Longrightarrow$ split into array, then process array.
- from left to right, in some other regular manner
  $\Longrightarrow$ substr() can be used instead of regular expression.
- by checking whether some pattern exists
  $\Longrightarrow$ 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);
```