Understanding Regular Expressions

Jakob Westhoff

Confoo 2012

About Me

Jakob Westhoff

About Me

Jakob Westhoff

About Me

Jakob Westhoff

- PHP Professional since 2001
- JavaScript Professional since 2006
- Trainer and Consultant
- Author of articles and a book
- Regular speaker at technology conferences

Qafoo
passion for software quality

Terminology

Terminology

RegExp

Subject

Match

Terminology

RegExp

- Pattern
  - Describes an arbitrary amount of strings
- Modifier
  - Processing instructions

Terminology

Subject

- Subject
  - One string which a RegExp is applied to

Terminology

Match

- Match
  - Part of the Subject which has been matched by the Regular Expression

Engine Flavors

Different RegExp Engines

- Different languages utilize different Regular Expression engines
  - PHP (PCRE)
  - Java
  - Python
  - Ruby
  - JavaScript
  - ...

Different RegExp Engines

- Different languages utilize different Regular Expression engines
  - PHP (PCRE)
  - Java
  - Python
  - Ruby
  - JavaScript
  - ...

RegExp

Basic structure of a RegExp

```
/foobar/i
```

Basic structure of a RegExp

- Delimiter
  - Enclosure of Pattern
  - Divider between Pattern and Modifier

Basic structure of a RegExp

```
(foobar)i
```

- Delimiter
  - PCRE allows arbitrary Brackets
    - `()` `[]` `{}`

RegExp – Just a String

- The RegExp Pattern is just a simple String

Techno

RegExp – Just a String

- The RegExp Pattern is just a simple String

```
(Techno)
```

RegExp – Just a String

- The RegExp Pattern is just a simple String

```
(Techno)
```

```
Web Techno Conference
```

RegExp – Just a String

- The RegExp Pattern is just a simple String

```
(Techno)
```

```
Web Techno Conference
```

  - The Pattern has to occur at least once
  - The Position inside the subject is not relevant

Metacharacters

- Certain characters inside a RegExp Pattern have got a special meaning

```
([We]b \s* Te+c.no)
```

Quantifier

Quantifier

- Quantifiers specify Repetitions of the previous character or group

```
(We*b Te+ch?n{1,3}o)
```

Quantifier

- Quantifiers specify Repetitions of the previous character or group

```
(We*b Te+ch?n{1,3}o)
```

  - `*` Any number of occurrences (0 → ∞)

Quantifier

- Quantifiers specify Repetitions of the previous character or group

```
(We*b Te+ch?n{1,3}o)
```

  - `*` Any number of occurrences (0 → ∞)
  - `+` One occurrence minimum (1 → ∞)

Quantifier

- Quantifiers specify Repetitions of the previous character or group

```
(We*b Te+ch?n{1,3}o)
```

  - * Any number of occurrences (0 → ∞)
  - + One occurrence minimum (1 → ∞)
  - ? Not at all or one time (0 → 1)

Quantifier

- Quantifiers specify Repetitions of the previous character or group

```
(We*b Te+ch?n{1,3}o)
```

  - `*` Any number of occurrences (0 → ∞)
  - `+` One occurrence minimum (1 → ∞)
  - `?` Not at all or one time (0 → 1)
  - `{x,y}` Between x and y (x → y)

The Dot

The Dot

- The Dot (`.`) matches any character
  - Everything except newline

```
(Make a .oint)
```

The Dot

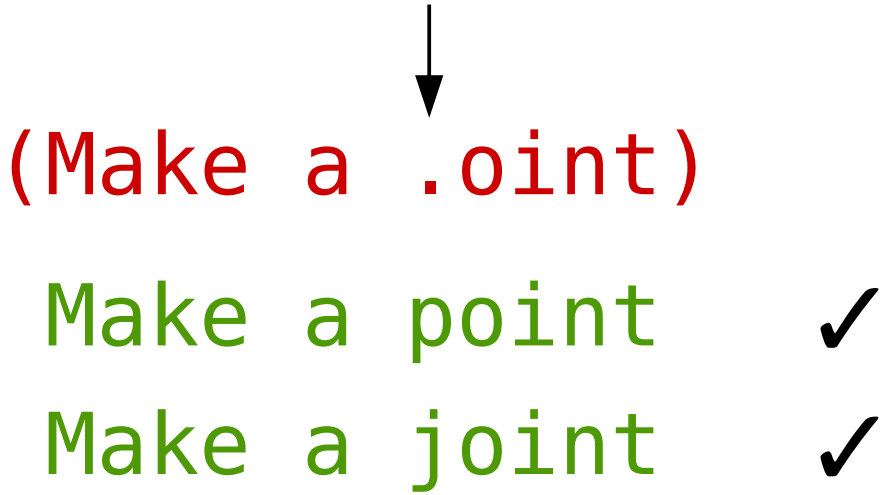
- The Dot (`.`) matches any character
  - Everything except newline


```
(Make a .oint)

 Make a point   ✓
```

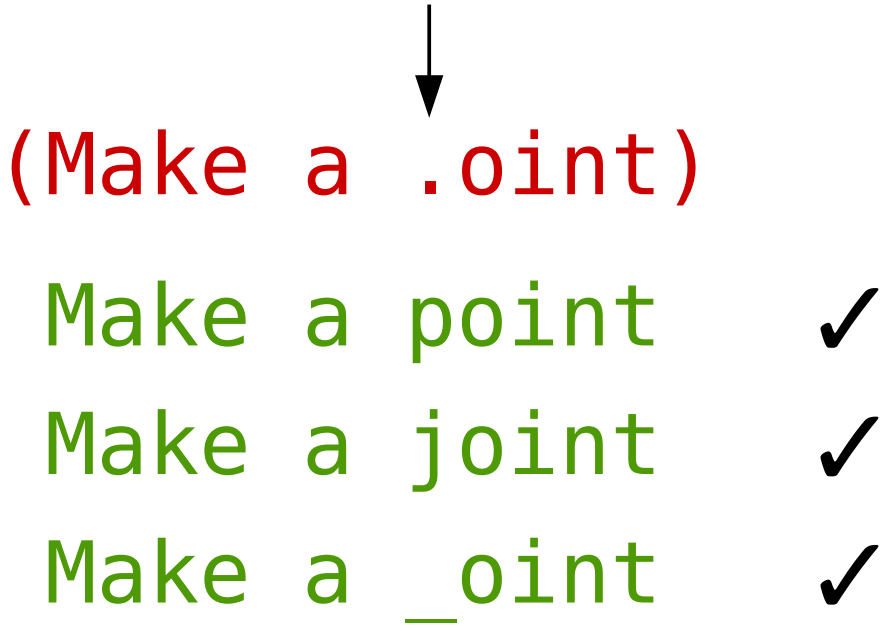
The Dot

- The Dot (`.`) matches <u>any</u> character
  - Everything except newline



The Dot

- The Dot (`.`) matches any character
  - Everything except newline



The Dot

- Switch to single line mode
  - Modifier s

```
(The.Dot)s
```

  - The Dot matches the newline character as well

The Dot

```
(The.Dot)s
```

- The Dot matches the newline character as well

The Dot

```
(The.Dot)s
```

- The Dot matches the newline character as well

```
The Dot
```
✓

The Dot

```
(The.Dot)s
```

- The Dot matches the newline character as well

```
The Dot    ✓
The:Dot    ✓
```

The Dot

```
(The.Dot)s
```

- The Dot matches the newline character as well

```
The Dot     ✓
The:Dot     ✓
The↵
Dot         ✓
```

Character Classes

Character Classes

- Character classes define a Set of arbitrary characters

a b c d e f

Character Classes

a b c d e f

Character Classes

```
abcdef
```

- No delimiters between characters

Character Classes

```
[abcdef]
```

- No delimiters between characters
- Enclosed by square brackets ([])

Character Classes

```
([abcdef]+)
```

- No delimiters between characters
- Enclosed by square brackets ([])
- Character Classes are treated as one character

Character Classes

```
([a-f]+)
```

- Ranges can be defined

Character Classes

```
([a-cd-f]+)
```

- Ranges can be defined
- One Character Class may contain multiple Ranges

Character Classes

```
([abc.]+)
```

- Metacharacters loose their special meaning

Character Classes

```
([abc.-]+)
```

- Metacharacters loose their special meaning
- New Metacharacters exist

Character Classes

```
([^abcdef]+)
```

- A Character Class can be negated

Character Classes

```
([^abcdef]+)
```

- A Character Class can be negated
- The newline character is part of the negation

Character Classes

```
([^\n]+)
```

- A Character Class can be negated
- The newline character is part of the negation
- The newline character can be excluded

Character Classes

- Predefined Character classes exist
  - `\d` Every digit (0,1,2,...)
  - `\s` Every whitespace (<Space>, <Tab>, ...)
  - ...
- Capitol letters negate the class
  - `\D` Everything but digits
  - ...

Alternatives

Alternatives

- Logical OR

```
(Open|Source)
```

Alternatives

- Logical OR

```
(Open|Source)
```

Open

Alternatives

- Logical OR

```
(Open|Source)
```

```
Open
```

✓

Alternatives

- Logical OR

```
(Open|Source)
```

```
Open      ✓
Source    ✓
```

Alternatives

- Logical OR

```
(Open|Source)
```

```
Open          ✓
Source        ✓
Open Source
```

Alternatives

- Logical OR

```
(Open|Source)
```

Open	✓
Source	✓
Open Source	✓

Escaping

Escaping

- Special meaning of Metacharacters can be disabled (Escaping)

jakob.westhoff@gmail.com

Escaping

- Special meaning of Metacharacters can be disabled (Escaping)

```
(jakob.westhoff@gmail.com)i
```

Escaping

- Special meaning of Metacharacters can be disabled (Escaping)

```
(jakob.westhoff@gmail.com)i
```

- This is a real dot not the Metacharacter, which represents any character

Escaping

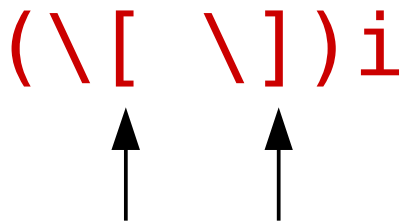
- Special meaning of Metacharacters can be disabled (Escaping)

```
(jakob\.westhoff@gmail\.com)i
```

- Using the Backslash to defuse Metacharacters (\)

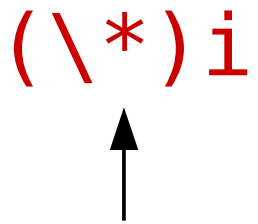
Escaping

- Works for any Metacharacter



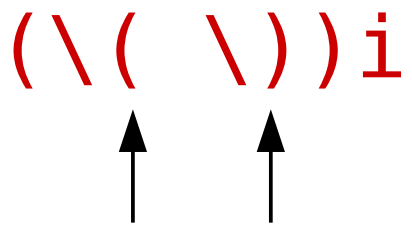
Escaping

- Works for any Metacharacter



Escaping

- Works for any Metacharacter



Escaping

- Works for any Metacharacter

```
(\+)i
```

Escaping

- Works for any Metacharacter

. . .

Escaping

- If you need a real backslash (\) you need to escape it as well

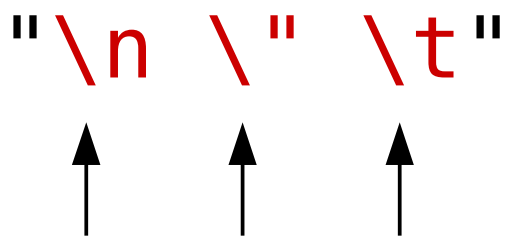
```
([a-z]+\\[0-9]+)i
```

Escaping in the real world

- Usually Regular Expressions are strings

Escaping in the real world

- Usually strings have their own escaping rules



Escaping in the real world

- Backslashes (\) in a Regular Expression string must be escaped themselves

```
"(jakob\\.westhoff@gmail\\.com)i"
```

Escaping in the real world

- What does this RegExp string match?

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- After the string escaping has been applied

```
('([^\\']+|\\\\|\\')+')
```

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- After the string escaping has been applied

```
('([^\\']+|\\\\|\\')+')
```

- Logical Or

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- After the string escaping has been applied

```
('([^\\']+|\\\\|\\')+')
```

- Character class containing everything but the backslash (\) or the single quote (')

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- After the string escaping has been applied

```
('([^\\']+|\\\\|\\')+')
```

- Two real backslashes (\\)

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- After the string escaping has been applied

```
('([^\\']+|\\\\|\\')+')
```

- Backslash (\) followed by a single quote (')

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- But what does it match?

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- But what does it match?

```
'A single quoted string,
with \'escaped\' single quotes and
\\backslashes\\'
```

Escaping in the real world

```
"('([^\\\\']+|\\\\\\\\|\\\\')+')"
```

- But what does it match?

```
'A single quoted string,
with \'escaped\' single quotes and
        \\backslashes\\'
```

Anchors

Anchors

- Anchors are part of the family of Assertions in Regular Expressions
- They are used to assert certain conditions without affecting the match
- Anchors: Beginning and end of the Subject

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

```
(Apple)i
```

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

```
(Apple)i
```

```
Apple
```

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

```
(Apple)i
```

```
Apple      ✓
```

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

```
(Apple)i
```

Apple ✓

Pineapple

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

```
(Apple)i
```

Apple ✓

Pineapple ✓

Anchors

- ^ Beginning of the Subject
- $ End of the Subject

```
(^Apple)i
```

Apple ✓

Pineapple ✘

Anchor (Modifier)

- Multiline Mode
  - Modifier m

```
(^abcdef$)m
```

- Anchors match the beginning and the end of each line inside the subject

Anchor (Modifier)

```
(^abcdef$)
```

```
abcdef↵
ghijkl↵
mnopqr
```

Anchor (Modifier)

```
(^abcdef$)
```

```
abcdef↵
ghijkl↵
mnopqr
```

- No match, as the anchors match the beginning and the end of the subject

Anchor (Modifier)

```
(^abcdef$)m
```

```
abcdef↵
ghijkl↵
mnopqr
```

- Anchors now match the beginning and end of every line inside the subject

Anchors

Multiline mode (m) independent anchors

- `\A` Beginning of subject
- `\z` End of subject

Anchor (Modifier)

- End only Mode
  - Modifier D

```
(^abcdef$)D
```

- $ only matches the “real” end of the subject
  - Usually a newline is allowed at the end of the subject

Subpattern

Subpattern

- Pattern can be divided using parenthesis

```
((abc)(def))
```

```
abcdef
```

Subpattern

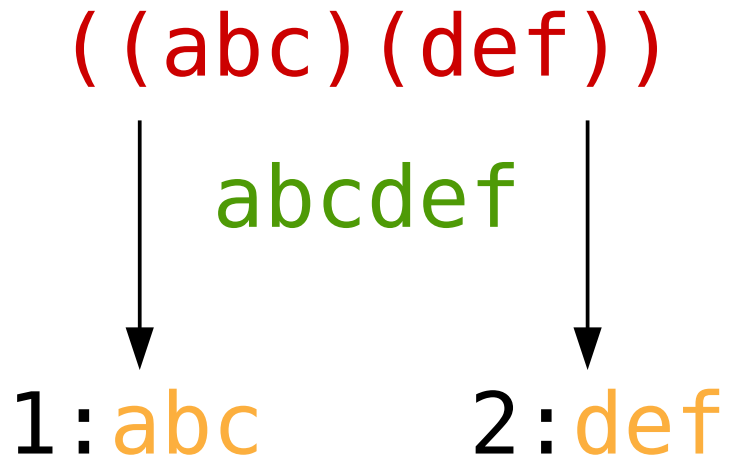
- Pattern can be divided using parenthesis

```
((abc)(def))
```

```
abcdef
```

Subpattern

- Pattern can be divided using parenthesis



  - Subpatterns may be used to extract parts of the match

Subpattern

```
((a.c)\1)
```

abcabc ✓

- Subpattern matches may be reused inside the pattern itself

Subpattern

```
((a.c)\1)
```

abcabc ✓

abcaXc ✘

- Subpattern matches may be reused inside the pattern itself

Subpattern Options

- Subpattern may be used to set options/modifiers for a certain area of the Regular Expression

```
((?#I am a comment subpattern.))
```

Subpattern Options

- Setting options for a subpattern

```
(?OptionPattern)
```

  - Abstract syntax for any option

Subpattern Options

- Setting the case-insensitive modifier using a subpattern option

```
((?i)[a-z]+)
```

Subpattern Options

- Setting the case-insensitive modifier using a subpattern option

```
((?i)[a-z]+)
```

Jakob Westhoff

Subpattern Options

- Setting the case-insensitive modifier using a subpattern option

```
((?i)[a-z]+)
```

```
Jakob Westhoff  ✓
```

Named Subpattern

- Subpatterns may be named

```
((?P<firstname>Jakob))
```

  - The P Option is used for naming subpatterns

Named Subpattern

```
((?P<firstname>Jakob) (Westhoff))
```

Named Subpattern

```
((?P<firstname>Jakob) (Westhoff))
```

Jakob Westhoff

Named Subpattern

```
((?P<firstname>Jakob) (Westhoff))
```

```
Jakob Westhoff
```

Named Subpattern

```
((?P<firstname>Jakob) (Westhoff))
```

Jakob Westhoff

```
firstname:Jakob
```

- Access to extraction using the subpatterns name is possible

Non grouping Subpattern

- Subpattern can be used without being a group

```
((?:Jakob))
```

  - The question mark followed by a colon (`?:`) creates a non grouping subpattern

Readability

Readable Regular Expressions

- Comments, indentation and line feeds in Regular Expressions
  - Modifier x

```
(foobar)x
```

Readable Regular Expressions

```
(^[a-z0-9_%.-]+@[a-z0-9.-]+\.[a-z]{2,4}$)iD
```

Easy to read? Easy to maintain?

Readable Regular Expressions

```
(
  ^                 #Start of the Subject
  [a-z0-9_%.-]+     #User
  @                 #Delimiter @
  [a-z0-9.-]+       #Domain
  \.                #Delimiter .
  [a-z]{2,4}        #TLD
  $                 #End of the Subject
)iDx
```

That's better :)

Readable Regular Expressions

```
(
  ^                 #Start of the Subject
  [a-z0-9_%.-]+     #User
  @                 #Delimiter @
  [a-z0-9.-]+       #Domain
  \.                #Delimiter .
  [a-z]{2,4}        #TLD
  $                 #End of the Subject
)iDx
```

- Use newlines where you see fit

Readable Regular Expressions

```
(
  ^                 #Start of the Subject
  [a-z0-9_%.-]+     #User
  @                 #Delimiter @
  [a-z0-9.-]+       #Domain
  \.                #Delimiter .
  [a-z]{2,4}        #TLD
  $                 #End of the Subject
)iDx
```

- Everything starting with a # until the end of line is considered a comment

Readable Regular Expressions

```
(
  ^                 #Start of the Subject
  [a-z0-9_%.-]+     #User
  @                 #Delimiter @
  [a-z0-9.-]+       #Domain
  \.                #Delimiter .
  [a-z]{2,4}        #TLD
  $                 #End of the Subject
)iDx
```

- All whitespaces are ignored if they are not escaped (`\ `)

Thanks for your attention.

https://joind.in/6075

Jakob Westhoff
Mail: jakob@qafoo.com
Twitter: @jakobwesthoff

Qafoo
passion for software quality