
Understanding Regular Expressions

the world of text processing

Jakob Westhoff <jakob@php.net>

PHP Unconference
September 12, 2009

About Me

- Jakob Westhoff
 - PHP developer for more than 6 years
 - Computer science student at the TU Dortmund
 - Co-Founder of the PHP Usergroup Dortmund
 - Active in different Open Source projects

Asking the audience

- Who does already work with regular expressions?

Asking the audience

- Who does already work with regular expressions?
- Regular expressions like this:

```
/[a-zA-Z]+/
```

Asking the audience

- Who does already work with regular expressions?
- Regular expressions like this:

```
/[a-zA-Z]+/
```

- Or like this:

```
(?P<image>(?:none|inherit)|(?:url\\(\\s*(?:'|")  
    ?(?:\\\\"'|\\\\"^\\'|\\\\"\\)\\)\\s*(?:'|")?  
    ))
```

Goals of this session

- **Writing** simple regular expressions
 - **Understand** the principles behind RegExp
 - Learn what **can** be done and what **can't** (or shouldn't)

Goals of this session

- **Writing** simple regular expressions
 - **Understand** the principles behind RegExp
 - Learn what **can** be done and what **can't** (or shouldn't)
- **Understanding** complex expressions
 - Learning to **write** complex RegExp needs **practice**
 - **Reading** them is not that hard

Goals of this session

- **Writing** simple regular expressions
 - **Understand** the principles behind RegExp
 - Learn what **can** be done and what **can't** (or shouldn't)
- **Understanding** complex expressions
 - Learning to **write** complex RegExp needs **practice**
 - **Reading** them is not that hard
- **PHP** and regular expressions
 - Howto use regular expressions in **PHP**
 - Some specialities of **PCRE**

What Regular Expressions are. . .

- In **theoretical** computer science:
 - Express **regular languages**

What Regular Expressions are. . .

- In **theoretical** computer science:
 - Express **regular languages**
 - Languages which can be described by **deterministic finite state automata**

What Regular Expressions are. . .

- In **theoretical** computer science:
 - Express **regular languages**
 - Languages which can be described by **deterministic finite state automata**
 - **Type 3 grammars** in the **Chomsky hierarchy**

What Regular Expressions are. . .

- In **practical** day to day usage:

“[. . .]regular expressions provide concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters.”

– Wikipedia [1]

What Regular Expressions are. . .

- In **practical** day to day usage:

“[. . .]regular expressions provide concise and flexible means for **identifying strings of text of interest**, such as particular characters, words, or patterns of characters.”

– Wikipedia [1]

What Regular Expressions are. . .

- In **practical** day to day usage:

“[. . .]regular expressions provide concise and flexible means for identifying strings of text of interest, such as **particular characters**, words, or patterns of characters.”

– Wikipedia [1]

What Regular Expressions are. . .

- In **practical** day to day usage:

“[. . .]regular expressions provide concise and flexible means for identifying strings of text of interest, such as particular characters, **words**, or patterns of characters.”

– Wikipedia [1]

What Regular Expressions are. . .

- In **practical** day to day usage:

“[. . .]regular expressions provide concise and flexible means for identifying strings of text of interest, such as particular characters, words, or **patterns of characters**.”

– Wikipedia [1]

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

- Delimiter

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

- Delimiter
 - **Equal** characters of arbitrary **choice** (must be escaped in expression)

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

- Delimiter
 - **Equal** characters of arbitrary **choice** (must be escaped in expression)
 - May be (and) in **PCRE**

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

- Delimiter
 - Equal characters of arbitrary choice (must be escaped in expression)
 - May be (and) in PCRE
- Expression

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

- Delimiter
 - Equal characters of arbitrary choice (must be escaped in expression)
 - May be (and) in PCRE
- Expression
- Modifier

Building Blocks of a Regular Expression

- Basic structure of every regular expression

`/[a-z]+/im`

- Delimiter
 - Equal characters of arbitrary choice (must be escaped in expression)
 - May be (and) in PCRE
- Expression
- Modifier
 - A sequence of characters providing processing instructions

Processing of a Regular Expression

- Regular expressions work **character-wise**

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Processing of a Regular Expression

- Regular expressions work **character-wise**
- All characters **without** a special meaning will be treated as a **simple character** match

Simple character-wise matching

```
/fo.bar/
```

```
foobar
```

Special characters

- . - Arbitrary character

Special characters

- . - Arbitrary character
- ^ - Start of subject (or line in multiline mode)

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)
- `(...)` - Subpattern grouping

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)
- `(...)` - Subpattern grouping
 - `(foo)(bar)`

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)
- `(...)` - Subpattern grouping
 - `(foo)(bar)`
- `|` - Logical Or

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)
- `(...)` - Subpattern grouping
 - `(foo)(bar)`
- `|` - Logical Or
 - `foo|bar`

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)
- `(...)` - Subpattern grouping
 - `(foo)(bar)`
- `|` - Logical Or
 - `foo|bar`

- Use special characters as normal ones by escaping them

Special characters

- `.` - Arbitrary character
- `^` - Start of subject (or line in multiline mode)
- `$` - End of subject (or line in multiline mode)
- `(...)` - Subpattern grouping
 - `(foo)(bar)`
- `|` - Logical Or
 - `foo|bar`
- Use special characters as normal ones by escaping them
 - `\.`, `\^`, `\$`, `\(`, ...

Character Classes

- [...] - Character classes

Character Classes

- [...] - Character classes
 - [abc]

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]
 - [^a-z]

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]
 - [^a-z]
- \d - Any decimal character

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]
 - [^a-z]
- \d - Any decimal character
- \D - Any **non** decimal character

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]
 - [^a-z]
- \d - Any decimal character
- \D - Any non decimal character
- \s - Any whitespace character

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]
 - [^a-z]
- \d - Any decimal character
- \D - Any **non** decimal character
- \s - Any whitespace character
- \S - Any **non** whitespace character

Character Classes

- [...] - Character classes
 - [abc]
 - [a-z]
 - [a-zA-Z_@-]
 - [^a-z]
- \d - Any decimal character
- \D - Any **non** decimal character
- \s - Any whitespace character
- \S - Any **non** whitespace character

- ... and some more: \h, \H, \v, \V, \w, \W

Quantifiers

- * - Zero or more occurrences

Quantifiers

- * - Zero or more occurrences
 - ab* - Matches: a, ab, abb, abbb, ...

Quantifiers

- * - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- + - **One** or more occurrences

Quantifiers

- * - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- + - **One** or more occurrences
 - `ab+` - Matches: `ab`, `abb`, `abbb`, ...

Quantifiers

- * - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- + - One or more occurrences
 - `ab+` - Matches: `ab`, `abb`, `abbb`, ...
- ? - None or one occurrence

Quantifiers

- * - Zero or more occurrences
 - `ab*` - Matches: a, ab, abb, abbb, ...
- + - One or more occurrences
 - `ab+` - Matches: ab, abb, abbb, ...
- ? - None or one occurrence
 - `ab?` - Matches: a, ab

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: a, ab, abb, abbb, ...
- `+` - **One** or more occurrences
 - `ab+` - Matches: ab, abb, abbb, ...
- `?` - **None** or **one** occurrence
 - `ab?` - Matches: a, ab
- `{min, max}` - Between **min** and **max** occurrences

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: a, ab, abb, abbb, ...
- `+` - One or more occurrences
 - `ab+` - Matches: ab, abb, abbb, ...
- `?` - None or one occurrence
 - `ab?` - Matches: a, ab
- `{min, max}` - Between min and max occurrences
 - `ab{2,4}` - Matches: abb, abbb, abbbb

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: a, ab, abb, abbb, ...
- `+` - One or more occurrences
 - `ab+` - Matches: ab, abb, abbb, ...
- `?` - None or one occurrence
 - `ab?` - Matches: a, ab
- `{min, max}` - Between min and max occurrences
 - `ab{2,4}` - Matches: abb, abbb, abbbb

- Can be applied to

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: a, ab, abb, abbb, ...
- `+` - One or more occurrences
 - `ab+` - Matches: ab, abb, abbb, ...
- `?` - None or one occurrence
 - `ab?` - Matches: a, ab
- `{min, max}` - Between min and max occurrences
 - `ab{2,4}` - Matches: abb, abbb, abbbb
- Can be applied to
 - simple characters

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- `+` - One or more occurrences
 - `ab+` - Matches: `ab`, `abb`, `abbb`, ...
- `?` - None or one occurrence
 - `ab?` - Matches: `a`, `ab`
- `{min, max}` - Between `min` and `max` occurrences
 - `ab{2,4}` - Matches: `abb`, `abbb`, `abbbb`
- Can be applied to
 - simple characters, the `.` special character

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- `+` - **One** or more occurrences
 - `ab+` - Matches: `ab`, `abb`, `abbb`, ...
- `?` - **None** or **one** occurrence
 - `ab?` - Matches: `a`, `ab`
- `{min, max}` - Between `min` and `max` occurrences
 - `ab{2,4}` - Matches: `abb`, `abbb`, `abbbb`
- Can be applied to
 - simple characters, the `.` special character, **character classes**

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- `+` - **One** or more occurrences
 - `ab+` - Matches: `ab`, `abb`, `abbb`, ...
- `?` - **None** or **one** occurrence
 - `ab?` - Matches: `a`, `ab`
- `{min, max}` - Between `min` and `max` occurrences
 - `ab{2,4}` - Matches: `abb`, `abbb`, `abbbb`
- Can be applied to
 - simple characters, the `.` special character, character classes, **subpattern groups**

Quantifiers

- `*` - Zero or more occurrences
 - `ab*` - Matches: `a`, `ab`, `abb`, `abbb`, ...
- `+` - One or more occurrences
 - `ab+` - Matches: `ab`, `abb`, `abbb`, ...
- `?` - None or one occurrence
 - `ab?` - Matches: `a`, `ab`
- `{min, max}` - Between `min` and `max` occurrences
 - `ab{2,4}` - Matches: `abb`, `abbb`, `abbbb`
- Can be applied to
 - simple characters, the `.` special character, character classes, subpattern groups, **back references**

Get a Feeling for Regular Expressions

- `/.*/`

Get a Feeling for Regular Expressions

- `./*/`
 - Any character *zero* or *many* times

Get a Feeling for Regular Expressions

- `./*/`
 - Any character *zero* or *many* times
- `/[a-z]+/`

Get a Feeling for Regular Expressions

- `/.*/`
 - Any character **zero** or **many** times
- `/[a-z]+/`
 - Any **lowercase** letter **one** or **many** times

Get a Feeling for Regular Expressions

- `/.*/`
 - Any character *zero* or *many* times
- `/[a-z]+/`
 - Any *lowercase* letter *one* or *many* times
- `/(foobar)*/`

Get a Feeling for Regular Expressions

- `./*/`
 - Any character **zero** or **many** times
- `/[a-z]+/`
 - Any **lowercase** letter **one** or **many** times
- `/(foobar)*/`
 - **Zero** or **many** times the string foobar (eg. [empty string], foobar, foobarfoobar, ...)

Get a Feeling for Regular Expressions

- `/.*/`
 - Any character *zero* or *many* times
- `/[a-z]+/`
 - Any *lowercase* letter *one* or *many* times
- `/(foobar)*/`
 - *Zero* or *many* times the string foobar (eg. [empty string], foobar, foobarfoobar, ...)
- `/foobar(\s*,\s*foobar)*/`

Get a Feeling for Regular Expressions

- `/.*/`
 - Any character **zero** or **many** times
- `/[a-z]+/`
 - Any **lowercase** letter **one** or **many** times
- `/(foobar)*/`
 - **Zero** or **many** times the string `foobar` (eg. [empty string], `foobar`, `foobarfoobar`, ...)
- `/foobar(\s*,\s*foobar)*/`
 - **One** or **many** **comma seperated** `foobar` strings with an **arbitrary amount** of whitespace characters between them

`/.*/`

foobar

- How many characters does this regular expression match?

`/.*/`

foobar

- How many characters does this regular expression match?
 - All of the characters? ("foobar")

`/.*/`

foobar

- How many characters does this regular expression match?
 - All of the characters? ("foobar")
 - **None** of the characters? ("")

`/.*/`

foobar

- How many characters does this regular expression match?
 - All of the characters? ("foobar")
 - None of the characters? ("")
- Depends on **greediness** of the regular expression

`/.*/`

foobar

- How many characters does this regular expression match?
 - All of the characters? ("foobar")
 - None of the characters? ("")
- Depends on **greediness** of the regular expression
 - Greedy (**default**)
 - Ungreedy

Switch greediness

- Switch greediness **globally**
 - Using a modifier

Switch greediness

- Switch greediness globally
 - Using a modifier
 - `/.*/U`

Switch greediness

- Switch greediness **globally**
 - Using a modifier
 - `/.*/U`
- Switch greediness **locally**
 - Using a quantifier followed by the `?` modifier

Switch greediness

- Switch greediness **globally**
 - Using a modifier
 - `/.*/U`
- Switch greediness **locally**
 - Using a quantifier followed by the `?` modifier
 - `/.*?/`

Switch greediness

- Switch greediness **globally**

- Using a modifier

- `/.*/U`

- Switch greediness **locally**

- Using a quantifier followed by the `?` modifier

- `/.*?/`

- `/.+?/`

Switch greediness

- Switch greediness globally
 - Using a modifier
 - `/.*/U`
- Switch greediness locally
 - Using a quantifier followed by the `?` modifier
 - `/.*?/`
 - `/.+?/`
 - `/.{2,4}?/`

Switch greediness

- Switch greediness **globally**
 - Using a modifier
 - `/.*/U`
- Switch greediness **locally**
 - Using a quantifier followed by the `?` modifier
 - `/.*/?`
 - `/.+?/`
 - `/.{2,4}?/`
- `?` modifier inverts greediness

Switch greediness

- Switch greediness **globally**
 - Using a modifier
 - `/.*/U`
- Switch greediness **locally**
 - Using a quantifier followed by the `?` modifier
 - `/.*?/`
 - `/.+?/`
 - `/.{2,4}?/`
- `?` modifier inverts greediness
 - `/.*?/` - Switch to **ungreedy** handling
 - `/.*?/U` - Switch to **greedy** handling

Subpatterns and Backreferences

- Why should one use subpatterns?

Subpatterns and Backreferences

- Why should one use subpatterns?
- Quantifiers
 - `/(foobar)*/`

Subpatterns and Backreferences

- Why should one use subpatterns?
- Quantifiers
 - `/(foobar)*/`
- Backreferences
 - `/(foo|bar)something(\1)/`

Subpatterns and Backreferences

- Why should one use subpatterns?
- Quantifiers
 - `/(foobar)*/`
- Backreferences
 - `/(foo|bar)something(\1)/`
 - Numbered by opening paranthesis

Subpatterns and Backreferences

- Why should one use subpatterns?
 - Quantifiers
 - `/(foobar)*/`
 - Backreferences
 - `/(foo|bar)something(\1)/`
 - Numbered by opening parenthesis
 - `/(foo(bar)(baz))/`

Subpatterns and Backreferences

- Why should one use subpatterns?

- Quantifiers

 - `/(foobar)*/`

- Backreferences

 - `/(foo|bar)something(\1)/`

 - Numbered by opening parenthesis

 - `/(foo(bar)(baz))/`

 - 1 foobarbaz

Subpatterns and Backreferences

- Why should one use subpatterns?

- Quantifiers

 - `/(foobar)*/`

- Backreferences

 - `/(foo|bar)something(\1)/`

 - Numbered by opening paranthesis

 - `/(foo(bar)(baz))/`

 - 1 foobarbaz

 - 2 bar

Subpatterns and Backreferences

- Why should one use subpatterns?

- Quantifiers

 - `/(foobar)*/`

- Backreferences

 - `/(foo|bar)something(\1)/`

 - Numbered by opening parenthesis

 - `/(foo(bar)(baz))/`

 - 1 foobarbaz

 - 2 bar

 - 3 baz

Subpatterns and Information Extraction

- Extracting information

```
preg_match( '/(foo(bar)(baz))/' , 'foobarbaz' , $matches );
```

Subpatterns and Information Extraction

- Extracting information

```
preg_match( '/(foo(bar)(baz))/' , 'foobarbaz' , $matches );
```

- Result

```
$matches = array(  
    0 => "foobarbaz" ,  
    1 => "foobarbaz" ,  
    2 => "bar" ,  
    3 => "baz" ,  
)
```

Grouping Without Subpattern Creation

- Grouping might be needed **without** creating a subpattern

```
/(?:foobar)*/
```

preg_* Functions in PHP

- `preg_match`
 - Match subject against a regular expression **once**

preg_* Functions in PHP

- `preg_match`
 - Match subject against a regular expression **once**
- `preg_match_all`
 - Match subject against a regular expression **as often as possible**

preg_* Functions in PHP

- `preg_match`
 - Match subject against a regular expression **once**
- `preg_match_all`
 - Match subject against a regular expression **as often as possible**
- `preg_replace`
 - Replace regular expression matches in a subject with something else

preg_* Functions in PHP

- `preg_match`
 - Match subject against a regular expression **once**
- `preg_match_all`
 - Match subject against a regular expression **as often as possible**
- `preg_replace`
 - Replace regular expression matches in a subject with something else
- `preg_split`
 - Split a string using a regexp as delimiter

preg_* Functions in PHP

- `preg_match`
 - Match subject against a regular expression **once**
- `preg_match_all`
 - Match subject against a regular expression **as often as possible**
- `preg_replace`
 - Replace regular expression matches in a subject with something else
- `preg_split`
 - Split a string using a regexp as delimiter
- `preg_quote`
 - Escape all special regexp characters in a given string

preg_* Functions in PHP

- `preg_match`
 - Match subject against a regular expression **once**
- `preg_match_all`
 - Match subject against a regular expression **as often as possible**
- `preg_replace`
 - Replace regular expression matches in a subject with something else
- `preg_split`
 - Split a string using a regexp as delimiter
- `preg_quote`
 - Escape all special regexp characters in a given string
- ... look at <http://php.net/manual/en/ref.pcre.php>

Digression About String Escaping

- Certain content needs to be escaped in PHP strings

Digression About String Escaping

- Certain content needs to be escaped in PHP strings
- In **double** quoted strings ("foobar")

Digression About String Escaping

- Certain content needs to be escaped in PHP strings
- In **double** quoted strings ("foobar")
 - \\$ - Dollar sign
 - \" - Double quote
 - \\ - Backslash

Digression About String Escaping

- Certain content needs to be escaped in PHP strings
- In **double** quoted strings ("foobar")
 - \\$ - Dollar sign
 - \" - Double quote
 - \\ - Backslash
 - All supported escape sequences: \n, \r, \t, \v, \x

Digression About String Escaping

- Certain content needs to be escaped in PHP strings
- In **double** quoted strings ("foobar")
 - \\$ - Dollar sign
 - \" - Double quote
 - \\ - Backslash
 - All supported escape sequences: \n, \r, \t, \v, \x
- In **single** quoted strings ('foobar')
 - \' - Single quote
 - \\ - Backslash

Digression About String Escaping

- Certain content needs to be escaped in PHP strings
- In **double** quoted strings ("foobar")
 - \\$ - Dollar sign
 - \" - Double quote
 - \\ - Backslash
 - All supported escape sequences: \n, \r, \t, \v, \x
- In **single** quoted strings ('foobar')
 - \' - Single quote
 - \\ - Backslash
- Advice
 - Use **single** quoted string to encapsulate regular expressions

Digression About String Escaping

- Certain content needs to be escaped in PHP strings
- In **double** quoted strings ("foobar")
 - \\$ - Dollar sign
 - \" - Double quote
 - \\ - Backslash
 - All supported escape sequences: \n, \r, \t, \v, \x
- In **single** quoted strings ('foobar')
 - \' - Single quote
 - \\ - Backslash
- Advice
 - Use **single** quoted string to encapsulate regular expressions
 - Even though it is not needed, **always** escape the backslash (\\)

String Escaping Example

Character class matching backslashes (`\`) and opening, as well as closing square brackets (`[]`)

String Escaping Example

Character class matching backslashes (\\) and opening, as well as closing square brackets ([])

```
"/[\\\\\\\\\\\\\\\\[\\]]*/"
```

String Escaping Example

Character class matching backslashes (\\) and opening, as well as closing square brackets ([])

```
"/[\\\\\\\\\\\\\\\\[\\]]*/"
```

String Escaping Example

Character class matching backslashes (\\) and opening, as well as closing square brackets ([])

```
"/[\\\\\\\\\\\\\\\\[\\]]*/"
```


String Escaping Example

Character class matching backslashes (\) and opening, as well as closing square brackets ([])

```
"/[\\\\\\\\\\\\\\\\[\\]]*/"
```

preg_match(_all) Examples

```
preg_match('(([A-Za-z]+)\s*)', 'foo_bar_baz',  
          $matches)
```

preg_match(_all) Examples

```
preg_match('(([A-Za-z]+)\\s*)', 'foo_bar_baz',  
          $matches)
```

■ Result

```
array (  
  0 => 'foo_bar_baz',  
  1 => 'foo_bar_baz',  
)
```

preg_match(_all) Examples

```
preg_match_all('([A-Za-z]+\s*)', 'foo_bar_baz',  
$matches);
```

preg_match(_all) Examples

```
preg_match_all('([A-Za-z]+\s*)', 'foo_bar_baz',  
    $matches);
```

■ Result

```
array (  
    0 =>  
        array (  
            0 => 'foo ',  
            1 => 'bar ',  
            2 => 'baz ',  
        ),  
    1 =>  
        array (  
            0 => 'foo ',  
            1 => 'bar ',  
            2 => 'baz ',  
        ),  
)
```

Subpattern Naming

- Remember: Subpatterns are numbered by their opening parenthesis position.

Subpattern Naming

- PCRE allows custom naming

```
/(?P<firstname>[A-Za-z]+) (?P<lastname>[A-Za-z]+)/
```

Subpattern Naming

- PCRE allows custom naming

```
/(?P<firstname>[A-Za-z]+) (?P<lastname>[A-Za-z]+)/
```

- Result with input Jakob Westhoff

```
array (  
  0 => 'Jakob_Westhoff',  
  'firstname' => 'Jakob',  
  1 => 'Jakob',  
  'lastname' => 'Westhoff',  
  2 => 'Westhoff',  
)
```

Assertions

- Formulate **assertions** on the matched string without consuming them

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

```
foo
```

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

```
foo
```

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

```
foofoo
```

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

```
foofoo
```

Assertions

- Formulate **assertions** on the matched string without consuming them
- Example

```
/foo(?=foo)/
```

- Input

```
foofoofoo
```

- Match

```
foofoo
```

Negative Assertions

- **Negative** assertions are possible

Negative Assertions

- **Negative** assertions are possible
- foo **not** followed by another foo

```
/foo(?!foo)/
```

Backward Assertions

- bar **preceeded** by foo

Backward Assertions

- bar **preceeded** by foo

`/(?=foo)bar/ ?`

Backward Assertions

- bar **preceeded** by foo

```
//(?!#foo)bar//
```

- Backward assertion

```
/(?<=foo)bar/
```

Backward Assertions

- bar preceded by foo

```
//(?!foo)bar//
```

- Backward assertion

```
/(?<=foo)bar/
```

- Negative backward assertion
- bar not preceded by foo

```
/(?<!foo)bar/
```

- Recursion in regular expressions ?

Pattern Recursion

- Recursion in regular expressions ?
- Possible with **PCRE**

Pattern Recursion

- Recursion in regular expressions ?
- Possible with PCRE
- Validate BB-Code using PCRE

```
[b>Hello [i]World[/i]![/b]
```

BB-Code Recursion Example

`[b]Hello [i]World[/i]![/b]`

- Recursive regular expression pattern

```
(  
  [^\[]*  
  \[(b|i)\]  
  (?:[^\[]+|(?R))  
  \[/\1\  
  [^\[]*  
)
```

BB-Code Recursion Example

`[b]Hello [i]World[/i]![/b]`

- Recursive regular expression pattern

```
(  
  [^\[]*  
  \[(b|i)\]  
  (?: [^\[]+ | (?R) )  
  \[/\1\  
  [^\[]*  
)
```

BB-Code Recursion Example

`[b]Hello [i]World[/i]![/b]`

- Recursive regular expression pattern

```
(  
  [^\[]*  
  \[(b|i)\]  
  (?:[^\[]+|(?R))  
  \[/\1\  
  [^\[]*  
)
```

BB-Code Recursion Example

`[b]Hello [i]World[/i]![/b]`

- Recursive regular expression pattern

```
(  
  [^\[]*  
  \[(b|i)\]  
  (?: [^\[]+ | (?R) )  
  \[/\1\  
  [^\[]*  
)
```

BB-Code Recursion Example

`[b]Hello [i]World[/i]![/b]`

- Recursive regular expression pattern

```
(  
  [^\[]*  
  \[(b|i)\]  
  (?: [^\[]+ | (?R) )  
  \[/\1\  
  [^\[]*  
)
```

BB-Code Recursion Example

`[b]Hello [i]World[/i]![/b]`

- Recursive regular expression pattern

```
(  
  [^\[]*  
  \[(b|i)\]  
  (?:[^\[]+|(?R))  
  [/\1\  
  [^\[]*  
)
```

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable
 - It is nearly impossible to find errors

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable
 - It is nearly impossible to find errors
 - Useful information extraction (building an AST) is not possible

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable
 - It is nearly impossible to find errors
 - Useful information extraction (building an AST) is not possible
- Use regular expressions for

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable
 - It is nearly impossible to find errors
 - Useful information extraction (building an AST) is not possible
- Use regular expressions for
 - Match Patterns (not recursive structures)

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable
 - It is nearly impossible to find errors
 - Useful information extraction (building an AST) is not possible
- Use regular expressions for
 - Match Patterns (not recursive structures)
 - Tokenizing strings

Do NOT Parse Using Regular Expressions

- Even though this is possible you do **NOT** want to do it
 - It is not maintainable
 - It is nearly impossible to find errors
 - Useful information extraction (building an AST) is not possible
- Use regular expressions for
 - Match Patterns (not recursive structures)
 - Tokenizing strings
 - Validate really restricted input values

Questions, comments or annotations?

Slides: <http://westhoffswelt.de/portfolio.htm>

Contact: Jakob Westhoff <jakob@php.net>

Bibliography I

- [1] Wikipedia.
Regular expressions — wikipedia, the free encyclopedia, 2002.
[Online; accessed 25-February-2002].