Perl6 Rules

vs. Perl5 regexp

Erik Johansen
DK Hostmaster A/S
2013-05-28

Regexp use

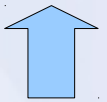
Some things stays the same

```
$ perl -E 'say "This is a Test" =~ /\w+/ && $&'
This

$ perl6 -e 'say "This is a Test" ~~ /\w+/'
｢This｣
```

Regexp use

Some things stays the same

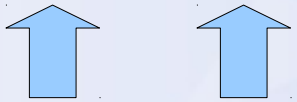
```
$ perl -E 'say "This is a Test" =~ /\w+/ && $&'
This

$ perl6 -e 'say "This is a Test" ~~ /\w+/'
「This」
```



Regexp use

Some things stays the same

```
$ perl -E 'say "This is a Test" =~ /\w+/ && $&'
This

$ perl6 -e 'say "This is a Test" ~~ /\w+/'
｢This｣
```



Regexp use

Some things stays the same

```
$ perl -E 'say "This is a Test" =~ /\w+/ && $&'
This

$ perl6 -
「This」
```

Character class short-cuts:

```
  \d \w \s \t \n \r \f \h \v
  \D \W \S \T \N \R \F \H \V

Quantifiers:
  .+          # 1..*
  .*          # 0..*
  .?          # 0..1
  .**1..10    # p5: .{1,10}
```

Regexp use

Swapping =~ and ~~

```
$ perl -E 'say "This is a Test" ~~ /\w+/ && $&'
This

$ perl6 -e 'say "This is a Test" =~ /\w+/'
===SORRY!===
Unsupported use of =~ to do pattern matching;
in Perl 6 please use ~~
at -e:1
------> say "This is a Test" =~    /\w+/
```

Regexp use

Swapping =~ and ~~

```
$ perl -E 'say "This is a Test" ~~ /\w+/ && $&'
This

$ perl6 -e 'say "This is a Test" =~ /\w+/'
===SORRY!===
Unsupported use of =~ to do pattern matching;
in Perl 6 please use ~~
at -e:1
------> say "This is a Test" =~    /\w+/
```

Regexp use

Capturing

```
$ perl -E 'say "This is a Test" =~ /(\w+)/'
This

$ perl6 -e 'say "This is a Test" ~~ /(\w+)/'
｢This｣
 0 => ｢This｣
```

Now starts at `$0, $1, $2...`

Regexp use

Capturing

```
$ perl -E 'say                                  /(\w+)/'
This

$ perl6 -e 'sa                                  ~ /(\w+)/'
「This」
 0 => 「This」
```

Non capturing brackets:

Perl5: `(?:\w+)`

Perl6: `[ \w+ ]`

Now starts at `$0, $1, $2...`

Regexp use

Result in $/

Perl6

```
"This is a Test" ~~ /(\w+)/;
```

Match result avaliable in `$/`

```
say $/.Bool;      # True

say $0;           # 「This」
say $/[0];        # 「This」

say ~$0;          # This
say ~$/[0];       # This
say $/[0].Str;    # This
```

Regexp use

Result in $/

Perl6

```
"This is a Test" ~~ / $<foo> = \w+ /;
```

Match result avaliable in `$/`

```
say $/.Bool;       # True
say $/<foo>;       # ｢This｣
say ~$/<foo>;      # This

say $/<foo>.Str;   # This
say $/<foo>.orig;  # This is a Test
say $/<foo>.from;  # 0
say $/<foo>.to;    # 4
```

Regexp use

Spaces
m//x by default.

```
$ perl -E 'say "This is a Test" =~ /(\w+ \w+)/'
This is

$ perl6 -e 'say "This is a Test" ~~
 m/(\w+ \w+)/'
｢This｣
 0 => ｢This｣
```

Regexp use

Spaces

Backwards compatibility mode

```
$ perl -E 'say "This is a Test" =~ /(\w+ \w+)/'
This is

$ perl6 -e 'say "This is a Test" ~~
 m:Perl5/(\w+ \w+)/'
｢This is｣
 0 => ｢This is｣
```

Regexp use

Spaces

Backwards compatibility mode

```
$ perl -E 'say "This is a Test" =~ /(\w+ \w+)/'
This is

$ perl6 -e 'say "This is a Test" ~~
 m:P5/(\w+ \w+)/'
｢This is｣
 0 => ｢This is｣
```

Regexp use

Spaces

Convert spaces into <ws> (almost like \s+)

```
$ perl -E 'say "This is a Test" =~ /(\w+ \w+)/'
This is

$ perl6 -e 'say "This is a Test" ~~
 m:sigspace/(\w+ \w+)/'
「This is」
 0 => 「This is」
```

Regexp use

Spaces

Convert spaces into <ws> (almost like \s+)

```
$ perl -E 'say "This is a Test" =~ /(\w+ \w+)/'
This is

$ perl6 -e 'say "This is a Test" ~~
 ms/(\w+ \w+)/'
｢This is｣
 0 => ｢This is｣
```

Regexp use

Spaces

Use <ws> or \s+

```
$ perl -E 'say "This is a Test" =~ /(\w+ \w+)/'
This is

$ perl6 -e 'say "This is a Test" ~~
 m/(\w+ <ws> \w+)/'
｢This is｣
 0 => ｢This is｣
```

Regexp use

Quoted string

```
#perl5
my $balltype = “foot”;

“football” ~~ /\Q${balltype}\Eball/;



#perl6
my $balltype = “foot”;

“football” ~~ / $balltype ball /;

“football” ~~ / $balltype “ball” /;
```

Regexp use

Quoted string

```
#perl5
my $balltype = “foot”;

“football” ~~ /\Q${balltype}\Eball/;
```

```
#perl6
my $balltype = “foot”;

“football” ~~ / $balltype ball /;

“football” ~~ / $balltype “ball” /;
```

Use `/ <$balltype> ball /` to interpolate.

Regexp use

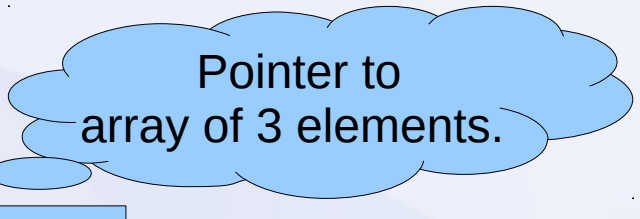
Quoted string

```
#perl5
my $balltype = “foot”;

“football” ~~ /\Q${balltype}\Eball/;



#perl6
my $balltype = “foot”;

“football” ~~ / $balltype ball /;

“football” ~~ / $balltype “ball” /;
```

Regexp use

Quoted string

```
#perl5
my $balltype = “foot”;

“football” ~~ /\Q${balltype}\Eball/;
```




```
#perl6
my $balltype = <foot hand walley>;
“football” ~~ / $balltype ball /;

“football” ~~ / $balltype “ball” /;
```

Regexp use

Modifiers

```
:i or :ignorecase – Ignore case
:g or :global     - More than once
:s or :sigspace   - Use whitespace as <ws>
:P5 or :Perl5     - Use Perl5 syntax
:ratchet          - Never perform backtracking
:c or :continue   - Continue from prev pos
:c(10) or :continue(10) – Continue from pos
:p(5) :pos(5)     - Match at exact pos
```

```
m:sigspace :ignorecase /\w+ \w+/
m:s:i/\w+ \w+/
ms/\w+ \w+/
ss:g/foo foo/bar/

m/ :i \w+/
```

Regexp use

Modifiers

```
:x(5)                  – Match number of times
:1x, :2x, :20x, ...

:nth(2)                - Get only Nth occurance
:1st, :2nd, :3rd, :4th, :5th, ...


m:3rd /\w+/;
m:3x /\w+/;
m:nth(1,3,4)/\w+/;
```

Regexp use

Match Start & End

```
$ perl -E 'say "This is a Test" =~ /^(\w+)/'
This

$ perl6 -e 'say "This is a Test" ~~ /^(\w+)/'
｢This｣
 0 => ｢This｣
```

Regexp use

Match Start & End

```
^       Start of string
$       End of string

^^      Start of any line in string
$$      End of any line in string

<<      Left side word boundary
>>      Right side word boundary
```

Grammar

Predefined rules.

```
<ident>     Match an identifier
            (alpha or underscore, followed by \w*)
<alpha>     Match a single alphabetic character
<digit>     Match a single digit (\d)
<alnum>     Match a single alphabetic or digit (\w)
<xdigit>    Match a single hexadecimal digit
<upper>     Match a single upper-case character
<lower>     Match a single lower-case character
<cntrl>     Match a single control character
<punct>     Match a single punctuation character

<prior>     Match using the most recent successful rule
```

Grammar

Predefined rules.

```
<ws>        Match any sequence of whitespace
<ww>        Match between to word characters (zero-width)
<wb>        Match word boundary (zero-width)
<space>     Match a single whitespace character
<blank>     Match a single space or tab


<?>         Match a null string. Always true.
<!>         Always false.
```

<.ws> Prefix with “.” to avoid default name capture.

Grammar

Predefined rules.

```
$ perl6 -e 'say "This is a Test" ~~ /<alnum>+/'
「This」
 alnum => 「T」
 alnum => 「h」
 alnum => 「i」
 alnum => 「s」

$ perl6 -e 'say "This is a Test" ~~ /<.alnum>+/'
「This」
```

Grammar

Predefined rules and naming.

```
$ perl6 -e 'say "open door" ~~ ms/<command=ident>
<object=ident>/'
「open door」
 ident => 「open」
 command => 「open」
 ident => 「door」
 object => 「door」

$ perl6 -e 'say "open door" ~~ ms/<command=.ident>
<object=.ident>/'
「open door」
 command => 「open」
 object => 「door」
```

Grammar

```
grammar URL {
    token TOP {
        <schema> '://'
        [ <ip> | <hostname> ]
        [ ':' <port>]?
        '/' <path>?
    }
    token schema   {  \w+                          }
    token ip       {  <byte> **  4     %  '.'      }
    token hostname {  (\w+)  **  2..*  %  '.'      }
    token port     {  \d+                          }
    token path     {
        <[ a..z A..Z 0..9 \-_.!~*'():@&=+$,/ ]>+
    }
    token byte     {  (\d**1..3) <{ $0 < 256 }>  }
}

my $url = 'http://perl6.org/documentation/';
my $match = URL.parse( $url );
say $match;
say "Hostname: ", $match<hostname>.Str;
# perl6.org
```

Grammar

```
grammar URL {
    token TOP {
        <schema> '://'
        [ <ip> | <hostname> ]
        [ ':' <port>]?
        '/' <path>?
    }
    token schema   {  \w+                          }
    token ip       {  <byte> **  4     %  '.'      }
    token hostname {  (\w+)  **  2..*  %  '.'      }
    token port     {  \d+                          }
    token path     {
        <[ a..z A..Z 0..9 \-_.!~*'():@&=+$,/ ]>+
    }
    token byte     {  (\d**1..3) <{ $0 < 256 }>  }
}

my $url = 'http://perl6.org/documentation/';
my $match = $url ~~ / <URL::TOP> /;
say $match;
say "Hostname: ", $match<URL::TOP><hostname>.Str;
# perl6.org
```

Grammar

```
my $g = grammar {
    token TOP {
        <schema> '://'
        [ <ip> | <hostname> ]
        [ ':' <port>]?
        '/' <path>?
    }
    token schema   {  \w+                          }
    token ip       {  <byte> **  4     %  '.'      }
    token hostname {  (\w+)  **  2..*  %  '.'      }
    token port     {  \d+                          }
    token path     {
        <[ a..z A..Z 0..9 \-_.!~*'():@&=+$,/ ]>+
    }
    token byte     {  (\d**1..3) <{ $0 < 256 }>  }
}

my $url = 'http://perl6.org/documentation/';
my $match = $g.parse( $url );      #  $g.parsefile($file)
say $match;
say "Hostname: ", $match<hostname>.Str;
# perl6.org
```

Grammar

Inherit grammar

```
grammar Letter {

    rule text  { <greet> <body> <close> }
    rule greet { [Hi|Hey|Yo] $<to>=(\S+?) , $$}
    rule body  { <line>+? }
    rule close { Later dude, $<from>=(.+) }
    # etc.
}

grammar FormalLetter is Letter {

    rule greet { Dear $<to>=(\S+?) , $$}
    rule close { Yours sincerely,
        $<from>=(.+) }

}
```

Grammar

```
grammar My::Grammar {
    regex re1 { … }
    token to2 { … }  # Implies :ratchet
    rule  ru3 { … }  # Implies :ratchet :sigspace
}
```

Looks and acts like Package + subs.
Accepts inheritance, and scoping with `my` and `our`.
Takes parameters.

Grammar

```
regex fruit { apple | orange | peach };


"This is a football test" ~~ regex
  { foo | football | handball };

say $/;  # football (longest match)
```

Strict order:

```
regex { foo || football || handball }
```

Grammar

Parameters

```
token quoted_str($q = '”') { $q .*? $q }

rule qq_str { <quoted_str('”')> }
rule q_str  { <quoted_str(“'”)> }
```

Grammar

```
regex ball {
    Football   |
    Handball   |
    Walleyball
}
```

Grammar

```
regex ball {
    | football
    | handball
    | walleyball
}
```

Grammar

```
regex command {
    | cp  : <file> <file>  { say “Copying” }
    | ls  : <file>         { say “Listing” }
    | cat : <file>         { say “Viewing” }
}
```

Grammar

Look ahead and look behind

```
"foobar" ~~ /<!after foo> bar/;  # Nil
"foobar" ~~ /<after foo> bar/;   # Match

"foobar" ~~ /foo <!before bar>/;  # Nil
"foobar" ~~ /foo <before bar>/;   # Match
```

Usually prefixed with '?' to prevent capturing:

```
<?after
<!?after
```

Modifiers

```
$_ = "foo\nBar\tBAZ";
say $_;

ss:i:samecase/foo bar baz/fee foo fum/;

say $_;
```

Output:

```
foo
Bar     BAZ
fee
Foo     FUM
```

:sigspace - Auto <.ws> seperator
:ignorecase :i - Ignore case
:samecase - Preserve case

Grammar

Set match goal.

```
$_ = "(foo,bar,baz),fluff";

/ "(" ~ ")"  <ident>+ % "," /;

say $/;
```

```
 (foo,bar,baz)
 ident =>  foo
 ident =>  bar
 ident =>  baz
```

Grammar

When match goal fails.

```
$_ = "(foo,bar,baz,fluff";

/ "(" ~ ")"  <ident>+ % "," /;


Unable to parse expression in ; couldn't find final ")"
  in any FAILGOAL at src/stage2/QRegex.nqp:1037
  in regex  at -e:1
  in method ACCEPTS at src/gen/CORE.setting:10413
  in method ACCEPTS at src/gen/CORE.setting:683
  in block  at -e:1
```

More info

`http://perlcabal.org/syn/S05.html`

A lot more features.

`http://www.uniejo.dk/presentations/2013-05-28.pdf`

This presentation

End of presentation...

End of presentation... Or....

Grammar

Not yet implemented in Rakudo.

Grammar

Predefined rules.

```
<self>      Match this entire pattern (recursively)
<sp>        Match a single space char
<null>      Match zero characters (i.e. unconditionally)
<ascii>     Match a single ASCII character
<ctrl>      Match a single control character
<graph>     Match a single non-control character
<print>     Match a single printable character
<word>      Same as \w
```

Grammar

Unicode properties.
Short-form and Long-form.

<L> or <Letter>	Match any letter
<Lu> or <UppercaseLetter>	Match any upper-case letter
<Sm> or <MathSymbol>	Match any mathematical symbol
<BidiWS>	Match any bidirectional whitespace
<Greek>	Match any Greek character
<Mongolian>	Match any Mongolian character
<Ogham>	Match any Ogham character
<Any>	Match any character
<InArrows>	Match any character in the "Arrows" block
<InCurrencySymbols>	Match any character in the "CurrencySymbols" block

etc.

Grammar

Match on already matched part.

```
"foo" ~~ /   <ident>  ~~ ^ f       /;

"foo" ~~ / [ <ident>  ~~ ^ f     ] /;

"foo" ~~ / [ <ident> !~~ ^ boo $ ] /;


"foo" ~~ / <ident>
        <?{ $<ident> !~~ / ^ boo $/ }> /;
```

Grammar

Allow partial match

```
regex ball {
    | f<*oot>b<*all>
    | h<*and>b<*all>
    | w<*alley>b<*all>
}
```

Grammar Grammar

Augmenting

```
use MONKEY_TYPING;

augment slang Regex {

    token backslash:sym<y> { ... }
    # define your own \y and \Y

    token assertion:sym<*> { ... }
    # define your own <*stuff>

    token metachar:sym<,> { ... }
    # define a new metacharacter

    multi method tweak (:$x) { ... }
    # define your own :x modifier

}
```

End of presentation...

End of presentation... O RLY ?

End of presentation... O RLY ? YA RLY !

More info

`http://perlcabal.org/syn/S05.html`

A lot more features.

`http://www.uniejo.dk/presentations/2013-05-28.pdf`

This presentation