COMP284 Scripting Languages

Lecture 5: Perl (Part 4)

Handouts

https://eduassistpro.github.i

Department of Computer
School of Electrical Engineering, Electronics, an
University of Liverpo

Contents

Assignment Project Exam Help

https://eduassistpro.github.i

Add WeChat edu_assist_pr

Regular expressions: Capture groups and backreferences

- We often encounter situations where we want to identify the repetition of the same or similar text, for example, in HTML markup:

```
<strong> ... </strong>
<li> ... </li>
```

- We mi
  but the

- We can
  using regular expressions:

```
<strong>.*<\/strong>
<li>.*<\/li>
```

but we cannot characterise both without losing fidelity, for example:

```
<\w+>.*<\/\w+>
```

does not capture the 'pairing' of HTML tags

Regular expressions: Capture groups

The solution are capture groups and backreferences

`(`*regexpr*`)`	creates a capture group
`(?<`*name*`>`*regexpr*`)`	creates a non-capturing group
`(?:`*regexpr*`)`	creates a non-capturing group
`\`*N*, `\g`	
`\g{`*na*	

Examples:

```
1  /Sorting took \d+\.\d+ seconds/
2  /<(\w+)>.*<\/\1>/
3  /([A-Z])0{2}(\d+)/
4  /(?<c1>\w)(?<c2>\w)\g{c2}\g{c1}/
5  /((?<c1>\w)(?<c2>\w)\g{c2}\g{c1})/
```

Regular expressions: Capture groups

Via capture variables the strings matched by a capture group are also available outside the pattern in which they are contained

$N	string matched by capture group $N (where N is a natural number)
$+{na	

The match...
code bloc

Example:

```
$_ = "... abba ... dato ... coo"
if (/((?<c1>\w)(?<c2>\w)\g{c2}\g{c1})/) {
  print "Match found: $1\n" }
```

Output:

```
Match found: abba
```

Regular expressions: Alternations

- The regular expression *regexpr1*|*regexpr2* matches
  if either *regexpr1* or *regexpr2* matches
  This type of regular expression is called an alternation

- Within a larger regular expression we need to enclose alternations
  in a capt
  (*reg

https://eduassistpro.github.i

Examples:

```
1 /Mr|Ms|Mrs|Dr/
2 /cat|dog|bird/
3 /(?:Bill|Hillary) Clinton/
```

Regular expressions: Alternations

- The order of expressions in an alternation only matters
  if one expression matches a sub-expression of another

Example:

```
1  $_ = "cats
2  if (/cat
3  if (/do
4  if (/(dog|dogs)/) { print "Match 3: $1\n" }
5  if (/(dogs|dog)/) { print "Match 4: $1\n" }
```

Output:

```
Match 1: cat
Match 2: cat
Match 3: dog
Match 4: dogs
```

Regular expressions: Anchors

Anchors allow us to fix where a match has to start or end

`\A`	Match only at string start
	Match only at string start (default)
	Match only at a line start (in `/.../m`)
`\Z`	
`\z`	
`$`	
`\b`	Match word boundary (between `\w`
`\B`	Match except at word boundary

Example:

```perl
$_ = "The girl who\nplayed with fire\n";
if (/fire\z/) { print "'fire' at string end\n" }
if (/fire\Z/) { print "'fire' at string end modulo \\n\n" }
```

```
'fire' at string end modulo \n
```

Regular expressions: Modifiers

Modifiers change the interpretation of certain characters in a regular
expression or the way in which Perl finds a match for a regular expression

/ /	Default
	'.' matches any character except '\n'
	'ˆ' matches only at string start
	'$' matches only at string end
/ /s	Treat string as single line
	'.' matches any character including '
	'ˆ' matches only at string start
	'$' matches only at string end
/ /m	Treat string as a set of multiple lines
	'.' matches any character except '\n'
	'ˆ' matches at a line start
	'$' matches at a line end

Regular expressions: Modifiers

Modifiers change the interpretation of certain characters in a regular expression or the way in which Perl finds a match for a regular expression

/ /sm	Treat string as a single long line, but detect multiple lines '.' matches any character including '\n'

| / /i | |

Example:

```
$_ = "bill\nclinton";
if (/(Bill|Hillary).Clinton)/smi) { print "Match:␣$1\n" }
```

Output:

```
Match: bill
Clinton
```

Regular expressions: Modifiers (/ /g and / /c)

Often we want to process all matches for a regular expression,
but the following code has not the desired effect

```perl
$_ = "11 22 33";
while (/\d+/) { print "Match starts at $-[0]: $&\n" }
```

The code a

```
Match starts a  0
```

To obtain
the / /g modifier:

/ /g	In scalar context, successive invocations ag
	move from match to match, keeping trac
	string
	In list context, returns a list of matched capture groups, or
	if there are no capture groups, a list of matches to
	the whole regular expression

Regular expressions: Modifiers (/ /g and / /c)

With the / /g modifier our code works as desired:

```perl
$_ = "11 22 33";
while (/\d+/g) { print "Match starts at $-[0]: $&\n" }
```

Output:

```
Match starts a  0
Match starts a  3
Match starts a  6
```

An example in a list context is the following:

```perl
$_ = "ab 11 cd 22 ef 33";
@numbers = /(\d+)/g;
print "Numbers: ", join(" | ", @numbers), "\n";
```

Output:

```
Numbers: 11 | 22 | 33
```

Read / /g as: Start to look for a match from the position where the last match using / /g ended

Regular expressions: Modifiers (/ /g and / /c)

The current position in a string for a regular expression *regexpr*
is associated with the string, not *regexpr*

↝ different regular expressions for the same strings will move forward the
   same position when used with / /g

↝ differ

   move f

Example

```
$_ = "ab␣11␣cd␣22␣ef␣33";
if (/\d+/g)    { print "Match␣starts␣at␣$-[0]:␣$&\n" }
if (/[a-z]+/g) { print "Match␣starts␣at␣$-[0]:␣$&\n" }
if (/\d+/g)    { print "Match␣starts␣at␣$-[0]:␣$&\n" }
```

Output:

```
Match starts at 3: 11
Match starts at 6: cd
Match starts at 9: 22
```

Regular expressions: Modifiers (/ /g and / /c)

A failed match or changing the target string resets the position

```
1  $_ = "ab 11 cd 22 ef 33";
2  if (/\d+/g) { print "2: Match starts at $-[0]: $&\n" }
3  if (/ab/g)  { print "3: Match starts at $-[0]: $&\n" }
4  if (/\d+/g) { print "4: Match starts at $-[0]: $&\n" }
```

Output:

```
2: Match start  a
4: Match start  a
```

To prevent the reset, an additional modifier

```
1  $_ = "ab 11 cd 22 ef 33";
2  if (/\d+/g)  { print "2: Match starts at $-[0]: $&\n" }
3  if (/ab/gc)  { print "3: Match starts at $-[0]: $&\n" }
4  if (/\d+/g)  { print "4: Match starts at $-[0]: $&\n" }
```

Output:

```
2: Match starts at 3: 11
4: Match starts at 9: 22
```

Generating regular expressions on-the-fly

The Perl parser will expand occurrences of $variable and @variable in regular expressions

⇒ regular expressions can be constructed at runtime

Example:

```
$_ = "Bart tea
@keywords = (
while ($keyw
  print "Match found for $keyword: $&\n" if /$keyword/i;
}
```

Output:

```
Match found for bart: Bart
Match found for lisa: Lisa
Match found for L\w+: Lisa
Match found for t\w+: teases
```

Binding operator

Perl offers two binding operators for regular expressions

string =~ /*regexpr*/	true iff *regexpr* matches *string*
string !~ /*regexpr*/	true iff *regexpr* does not match *string*

- Note th

- Most o
  return t
  variab

Example:

```
$name = "Dr␣Ullrich␣Hustadt";
if ($name =~ /(Mr|Ms|Mrs|Dr)?\s*(\w+)/) {print "H
$name = "Dave␣Shield";
if ($name =~ /(Mr|Ms|Mrs|Dr)?\s*(\w+)/) {print "Hello␣$2\n"}
```

```
Hello  Ullrich
Hello  Dave
```

Pattern matching in a list context

- When a pattern match /*regexpr*/ is used in a list context, then the return value is
  - a list of the strings matched by the capture groups in *regexpr* if the match succeeds and *regexpr* contains capture groups, or
  - (a list
    if the
  - an e

```perl
$name = "Dr␣Ullrich␣Hustadt";
($t,$f,$l) = ($name =~ /(Mr|Ms|Mrs|Dr)?
print "Name:␣$t,␣$f,␣$l\n";
$name = "Dave␣Shield";
($t,$f,$l) = ($name =~ /(Mr|Ms|Mrs|Dr)?
print "Name:␣$t,␣$f,␣$l\n";
```

Output:

```
Name: Dr, Ullrich, Hustadt
Name: , Dave, Shield
```

Pattern matching in a list context

- When a pattern match /*regexpr*/g is used in a list context, then the return value is

  - a list of the strings matched by the capture groups in *regexpr* each time regex matches
    provided that *regexpr* contains capture groups, or

  - a list c
    mat

  - an e

```
$string = "firefox:␣10.3␣seconds;␣chrome:␣9.5␣seconds";
%performance = ($string =~ /(\w+)\:\s
foreach $system (keys %performance) {
    print "$system␣->␣$performance{$system}\n" }
```

Output:

```
firefox -> 10.3
chrome -> 9.5
```

Revision

Read

- Chapter 7: In the World of Regular Expressions
- Chapter 8: Matching with Regular Expressions

of

R. L. Sch

Learning Perl.

O'Reilly 2011

- `http://perldoc.perl.org/perlre.html`
- `http://perldoc.perl.org/perlretut.html`
- `http://www.perlfect.com/articles/regextutor.shtml`