Regular Expressions in Perl

Pat Morin

COMP 2405

Carleton
UNIVERSITY
Canada's Capital University

The =~ Operator

- In Perl, we can use regular expressions to match (parts of) strings
- This is done with the =~ operator
- This operator evaluates to true if the expression matches the string and false otherwise
- Note that the text between the / and / is processed as a double-quoted string

```
if ($line =~ /__FILENAME__/) {
  print("This line contains __FILENAME__\n");
}
```

What is a regular expression?

- The simplest regular expressions are just strings
- This is the case if the RE doesn't contain any special characters such as `\^.$|()[]{}*+?,/`
- If you want to include a special character, just escape it with `\`

```
if ($formula =~ /e=mc\^2/) { .... }
```

Carleton UNIVERSITY
Canada's Capital University

The Alternation Operator

- The choice | operator creates a regular expression that matches one of two things

```
if ($lastName =~ /Morin|Lebowsky/) {
   PrepareWhiteRussian();
}
```

The Character Class Operator

- The character class operator [] allows to match any character within the class
- `[abcdefg]` is equivalent to `a|b|c|d|e|f|g`
- There are a few predefined character classes
  - `.` is (almost) any character
  - `\d` any digit 0-9 `[0123456789]`
  - `\s` any whitespace `[ \t\n\r]`
  - `\w` any *word* character `[_0123456789abc...zABC...Z]`

```
if ($lastName = /[Ll1]ebowsky/) { ... }
if ($year = /\d\d/) { ... }
if ($isoDate = /\d\d\d\d-\d\d-\d\d/) { ... }
```

Carleton UNIVERSITY
Canada's Capital University

More Character Classes

- There are lots of predefined character classes
  - \W any non-word character
  - \S any non-space character
  - \D any non-digit character
- There are also POSIX character classes
- Syntax is `[:class:]` where class is one of
  - `alpha`, `alnum`, `ascii`, `blank`, `cntrl`, `digit`, `graph`, `lower`, `print`, `punct`, `space`, `upper`, `word`, `xdigit`.

The Range Operator

- The range operator {} can be used to match the same expression repeatedly
  - * match 0 or more times
  - + match 1 or more times
  - ? match 1 or 0 times
  - `{n}` match exactly n times
  - `{n,}` match at least n times
  - `{n,m}` match at least n but not more than m times
- These operators are *greedy*, they will try to match as many as possible

The Range Operator - Examples

```
if ($year =~ /\d{4}/) { ... }
if ($date =~ /\d{4}-\d{1,2}-\d{1,2}/) { ... }
if ($word =~ /\w+/) { ... }
if ($oneOrMany =~ /\sabstract(s?)\s/) { ... }
if ($numbers =~ /((\d+)\s+)*/) { ... }
```

Carleton UNIVERSITY
Canada's Capital University

Special Characters

- There are some important special characters
  - ^ is the beginning of the string
  - $ is the end of the string (or before the newline)
- You can use ^ and $ to make sure your strings don't contain garbage
  - This is good practice for validating user input

Carleton UNIVERSITY
Canada's Capital University

```
if ($filename =~ /^\w*$/) {
  print("$filename is a safe filename");
}
```

Modifiers

- Modifiers that appear after the second / control aspects of the RE matching process
  - `i` makes the RE case-insensitive
  - `x` ignores whitespace in the RE (for readability)
  - `m` treats string as a multiline string
  - `s` treats string as single line string (so . matches `\r` and `\n`)
  - `g` performs a global search (for repeated searches)

```
$date =~ /\d{4} -
          \d{1,2} -
          \d{1,2}/x;
```

Carleton UNIVERSITY
Canada's Capital University

The Return Value

- The =~ operator actually returns a value
- If no parentheses appear in the RE then =~ returns true or false (1 or 0)
- If parentheses appear in the RE then =~ returns an array whose elements are the parenthesized parts of the expression that match

```
$year = ($line =~ /\d{4}/);

($lastName, $firstName)
  = ($input =~ /\s*(\w+)\s*\,\s*(\w+)\s*$/);
```

Carleton UNIVERSITY
Canada's Capital University

Substitutions

- The RE mechanism also allows for substitutions using the `s///` operator
- Usage: `s/`*`pattern`*`/`*`replacement`*`/`
- The g modifier allows substitution of all occurrences of *pattern* with *replacement* (otherwise only the first is substituted)
- **Note:** These modify the string

```
$line =~ s/__TITLE__/My Document Title/;
$line =~ s/__USERNAME__/$username/;
$leet =~ s/[Ll]/1/g;
$leet =~ s/[Ee]/3/g;
```

Carleton UNIVERSITY
Canada's Capital University

Summary

- Perl regular expressions allow us to
  - Search for patterns in strings
  - Handle user input in a robust way
  - Substitute portions of strings with other strings
- The pattern is treated like a double-quoted string (with variable substitution and escape characters)
- The parts of the pattern can be returned as an array (when using parentheses)
- The substitution operator is a great way to manipulate text
- See Chapter 9 of Perl 5 Tutorial for more info.