Learning Ruby

Regular Expressions

Get at practice page by logging on to csilm.usu.edu and selecting
PROGRAMMING LANGUAGES|Regular Expressions

Regular Expressions

• A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern

• You use them in
  – asking if a string contains certain features
  – extracting words which fit the given patterns from the string
Uses of regular expressions

- Used to deal with needs such as
  - Does this string contain something of the form GPA: followed by digits followed by a slash
  - if string contains a date of the form mm/dd/yyyy, extract month, day and year.
  - Extra all lines that begin with “money” or “scandall”
  - Give me a list of all identifiers in a string where identifiers begin with a alpha character or @, contain only letters, digits, and underscores, but may end with a ? or !.

- Regular expressions are available in many languages (AWK, Sed, Perl, Python, C/C++)
- Matching strings with RegExp’s is very efficient and fast
- In Ruby, RegExp’s are objects, like everything else

Components of Regular Expressions

- literals - match exactly what I give you
- character groups - match a letter of one of these
- anchors - the string must begin or end with the match
- counts – repeat pattern in the following way
- groupings – group parts of the match to extract as one piece
- alternations – match this or that
**Literals**

Most characters match themselves.

\[=\sim\] is the pattern match operator

pattern is enclosed in slashes

\[s = "I am really having fun today"
\]

if \(s = \sim /\text{really}/\)

\[p \text{ "Please don't use really "}
\]

end

You may need to backslash special characters to match them exactly.

\[a = "\text{googleocom google.com}"
\]

\[p \text{ showRE(a, /google\.com/) -> } "\text{googleocom} >>\text{google.com}<<"
\]

---

**Character Groups**

Written between square brackets, these match one character from test string against one character from the group.

\[a = "I expect to be paid in $ or EU"
\]

\[p \text{ showRE(a, /[@$]>/) -> } "\text{I expect to be paid in} >>$<< \text{ or EU}"
\]

\[a = "\text{Old Mac Donald Had a cow eieio}"
\]

\[p \text{ showRE(a, /[aeiou]{2,}/) -> } "\text{Old Mac Donald Had a cow} >>\text{eieio}<<"
\]
Character Groups (cont)

• If ^ is the first character after [, you match anything EXCEPT what is in the character group.
• /p[aeiou][^d]/ matches a p followed by a vowel followed by anything which is not a lower case d. matches pat, pen, but not pad
• ‘’ matches any one character (but must be something)
  /[^hd]ee./ matches peet, weed, but not tee or heet
• You can use hyphen within a character group
  /P[a-z][0-9]/

– Inside the character group, most characters are treated literally (not their special meaning)
– In the character group, you can use the literal characters ] and – if they appear first: []-abn-z
Predefined character classes

- These work inside or outside []’s:
  - \d = digit = [0-9]
  - \D = non-digit = [^0-9] (inverted digit)
  - \s = whitespace,
  - \S = non-whitespace (inverted whitespace)
  - \w = word character [a-zA-Z0-9_]
  - \W = non-word character (inverted word)

Capital version is always NO, as in “DIDN’T YOU HEAR ME? I SAID NO DIGITS.”

Anchors

- By default, regular expression matches are made anywhere within the test string
- Anchors: you use ^ in the pattern to indicate "start of string" and $ to indicate end of the test string.
  For example - then you can limit you match to the start or end.
- Note, ^ inside [] is not an anchor.
- if you do both beginning and ending anchor, you're specifying a regular expression that matches the whole string.
- /CAT/ strings that begin with CAT
- /ONLY THIS$/ strings that of exactly “ONLY THIS”
- a= "The cow is a cow is a cow"
- p showRE(a, /cow$/) -> "The cow is a cow is a >>cow<<"
Counts

• Each literal, character group (and anchor) that you've seen so far tries to matches once against the test string.
• By adding a count \{min,max\} AFTER any of these elements, you can specify that you want it to match a different number of times.
  \[a= "Yodelling is fun ooiiioeeioe."\]
  \[p \text{showRE}(a, /[aeiou]{3,}/) \rightarrow "Yodelling is fun >>ooiiioeeioe<<."\]
• Besides specific counts, we have other counts:
  ? previous item occurs 0 or 1 times ("perhaps a")
  + previous item occurs 1 or more times ("some")
  * previous item occurs 0 or more times ("perhaps some")
• /ha*t/
• /^[aeiou]+[a-zA-Z0-9]?/
Groupings

• If you want your counts to apply to more than one character, you can use parentheses around the section to which the count applies
• /Very Funny (ho)+/

Alternation

• The "|" character in a regular expression means "or"
• Or is low precedence – alternates entire regexps unless grouped
  – /red ball|angry sky/ matches “red ball” or “angry sky” not “red ball sky” or “red angry sky”
  – /red (ball|angry) sky/ does the latter
• / [http][ftp]/ would match ht, hp, tt, ...
• /(http|ftp)/ would match http or ftp
• /(h|H)(e|E)(l|L)(p|P)/ matches HELP, HeLP,...
Captures – extracting matched parts

- having matched, you often want to refer to the part of the test_string that matched specific parts of the regular expression.
  - What was the month?
  - I want to remember something to use it again (e.g., we want beginning and ending tags in html to match)
- What can be referred to must be in a group.
- In order to capture part of the incoming string, you should use a set of grouping brackets to indicate the 'interesting part'.

Captures - backreferencing

- After you match a regular expression some “special” Ruby variables are automatically set:
  - $& is the part of the expression that matched the pattern
  - $` ($forward quote)the part of the string before the pattern
  - $’ ($backquote) the part of the string after the pattern
When you use ()’s for grouping, Ruby assigns the match within the first () pair to:

- \1 (if you are using it within the pattern - so you can refer to it again)
- $1 outside the pattern

“mississippi” =~ /.*(iss)+.*$/ » $1 = “iss”

vowel = "peak at a door. steep rhymes with deep leak"
if vowel =~ /([aeiou][aeiou]).*\1/ repeating vowel pair
  puts "matches", $1 → ea
end

Only remembers things in parens
If there are two sets of parens, they are numbered sequentially

while line = gets  # not null
  line =~ /\w+ (\w+) \w+ (\w+)/
  puts "Second word: '#{$1}' on line #{$.}" if defined? $1
  puts "Fourth word: '#{$2}' on line #{$.}" if defined? $2
end # of while.

$. is incremented every time you do a gets
Parens can be nested. Numbered as corresponding to left paren order. (job #\d*(\d+))
• Checking for double words: \b is word boundary
  /\b(\w+)\s+\1\b/
• Checking for triple characters
  /(.\)\1\1/  
• Match anything between <> other than a >
  /<[^>]*>/
• find an email subject in a string and pass the value to a variable called 'subject'.
  email = "Subject: why I love Ruby"
  email =~ /^Subject:\ (.+)$/
  subject = $1  ← why I love Ruby

Helps in understanding reg expression

def showRE( target, regex )
  if target =~ regex
    # Highlight with chevrons where match occurs.
    "#{\$`}>>#{\$&}<<#{\$'}"
  else
    "no match found"
  end  # of if.
end # of show_regexp.
Repetition and greediness

• By default, repetition is *greedy*, meaning that it will assign as many characters as possible.

• You can make a repetition modifier non-greedy by adding ‘?’

```javascript
a = "The moon is made of cheese"
```

```
showRE(a, /\w+/) → ">>The<< moon is made of cheese"
showRE(a, /\s.*\s/) → "The>> moon is made of <<cheese"
showRE(a, /\s.*?\s/) → "The>> moon <<is made of cheese"
showRE(a, /[aeiou]{2,99}/) → "The m>>oo<<n is made of cheese"
showRE(a, /mo?o/) → "The >>moo<<n is made of cheese"
showRE(a, /mo??o/) → "The >>mo<<on is made of cheese"
```

```
string = "<h3>My Title</h3> <h3> my Rank </h3>"
```

```javascript
p showRE(string, /\<h[0-9]\>.*?\</h[0-9]\>/)
-→">>><h3>My Title</h3>>< <h3> my Rank </h3>“
```

with greedy, we get
-→ ">>><h3>My Title</h3> <h3> my Rank </h3>"
String Methods Revisited

- `s.split(regexp)` – returns a list of substrings, with regexp as a delimiter
  - Can assign to an array, or use multiple assignment

```ruby
oldies| 3.5 | Beatles | I want to hold your hand
line = "oldies| 3.5 | Beatles | I want to hold your hand"
file, length, name, title = line.chomp.split(/\s*\|\s*/)
puts "(" + file + ") (" + length + ") (" + name + ") (" + title + ")"

->(oldies) (3.5) (Beatles) (I want to hold your hand)
```

Squeeze

- `s.squeeze(reps)` – reduces in s any runs of more than one character from string reps to only one

```ruby
line = "OOOH HECKKKK why??"
reps = "O?"
p "#{line.squeeze(reps)}"
  ->" OH HECKKKK why?"
```
String Methods

• s.scan(regexp) – returns a list of parts that match the pattern

```ruby
st = "123 45 hello out67there what's 23up?"
digs = st.scan(/\d+/)
p digs
  -> ["123", "45", "67", "23"]
```

Many more in Built-in Classes and Methods!

Regexp substitutions

• a.sub (one replacement)

```ruby
subject = "before the world and before the fire there were dinosaurs"
result = subject.sub("before", "after")
p result
  -> "after the world and before the fire there were dinosaurs"
```

• a.gsub (global)

```ruby
s = "My hovercraft and hovercraft compartment are full of eels"
s.gsub!/hovercraft/, 'spaceship')
  "My spaceship and spaceship compartment are full of eels"
```

• Can use captures, but form is different

```ruby
s = "Billy dog dog is my favorite favorite animal animal"
t = s.gsub!/\w*\?\1/ {1}
  Billy dog is my favorite animal
```
Regexp substitutions

• Using back references
  
  \text{string1} = "I will drill for a well in walla walla washington."
  \text{string1.gsub!}(/(w.ll)/){$1.uppercase} \leftarrow \text{replaces each differently}
  \text{puts string1} \rightarrow \text{I WILL drill for a WELL in WALLa WALLa washington.}
  
  \text{string2} = "I will drill for a well in Logan Utah will you?."
  \text{string2.gsub!}(/(w.ll)/){$1.uppercase}
  \text{puts string2} \rightarrow \text{I WILL drill for a well in Logan Utah will you?}.

• Backreference within pattern
  
  \text{s} = "My hovercraft is full of eels"
  \text{s.gsub!}(/(hovercraft)\1/", "HOPE")
  \text{puts s} \rightarrow \text{My HOPE is full of eels}

There is a nice way to convert a string to an integer if it is in fact an integer, and to return nil if it’s not.

\begin{verbatim}
  def safe_string_to_int(string)
    if /^\d+$/match(string)
      string.to_i(10)
    else
      nil
    end
  end
\end{verbatim}
Can read in or create regular expressions via execution

puts "enter your regular expression"
reg = gets #enter Subject .* (\w+)$
reg.chomp!
pattern = Regexp.new(reg)
showRE(email, pattern)