Regular Expression

Outline

1. Introduction
2. Expression Modifiers
3. Expression Patterns
4. Search and Replace

1. Introduction

- 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.
- A *regular expression literal* is a pattern between slashes or between arbitrary delimiters followed by %r as follows

```
/pattern/
/pattern/im    # option can be specified
%r!/usr/local! # general delimited regular expression
```

```
#!/usr/bin/ruby
line1 = "Cats are smarter than dogs";
line2 = "Dogs also like meat";
puts "Line1 contains Cats" if ( line1 =~ /Cats(.*)/ )
puts "Line2 contains  Dogs" if ( line2 =~ /Cats(.*)/ )
```


2. Expression Modifiers

Regular expression literals may include an optional modifier to control various aspects of matching. The modifier is specified after the second slash character, as shown previously and may be represented by one of these characters

2. Expression Modifiers

No.	Modifier & Description
1	**i** Ignores case when matching text.
2	**o** Performs #{} interpolations only once, the first time the regexp literal is evaluated.
3	**x** Ignores whitespace and allows comments in regular expressions.
4	**m** Matches multiple lines, recognizing newlines as normal characters.
5	**u,e,s,n** Interprets the regexp as Unicode (UTF-8), EUC, SJIS, or ASCII. If none of these modifiers is specified, the regular expression is assumed to use the source encoding.

2. Expression Modifiers

Like string literals delimited with %Q, Ruby allows you to begin your regular expressions with %r followed by a delimiter of your choice. This is useful when the pattern you are describing contains a lot of forward slash characters that you don't want to escape

```
# Following matches a single slash character, no escape required
%r|/|

# Flag characters are allowed with this syntax, too
%r[</(.*)>]i
```


3. Expression Patterns

Except for control characters, (+ ? . * ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash.

The regular expression syntax that is available in Ruby, pls following references:

- http://rubular.com/
- http://regexr.com/
- https://regex101.com/

Example:

^ => Matches beginning of line.

$ => Matches end of line.

. => Matches any single character except newline. Using m option allows it to match newline as well.

[...] => Matches any single character in brackets.

4. Search and Replace

- Some of the most important String methods that use regular expressions are **sub** and **gsub**, and their in-place variants **sub!** and **gsub!**.
- All of these methods perform a search-and-replace operation using a Regexp pattern. The **sub** & **sub!** replaces the first occurrence of the pattern and **gsub** & **gsub!** replaces all occurrences.
- The **sub** and **gsub** returns a new string, leaving the original unmodified where as sub! and gsub! modify the string on which they are called.

References:

https://ruby-doc.org/core-2.1.4/String.html#method-i-gsub

https://ruby-doc.org/core-2.1.4/String.html#method-i-gsub-21

https://ruby-doc.org/core-2.1.4/String.html#method-i-sub

https://ruby-doc.org/core-2.1.4/String.html#method-i-sub-21

4. Search and Replace (cont.)

```
#!/usr/bin/ruby

phone = "2004-959-559 #This is Phone Number"

# Delete Ruby-style comments
phone = phone.sub!(/#.*$/, "")
puts "Phone Num : #{phone}"

# Remove anything other than digits
phone = phone.gsub!(/\D/, "")
puts "Phone Num : #{phone}"
```


4. Search and Replace (cont.)

```
#!/usr/bin/ruby

text = "rails are rails, really good Ruby on Rails"

# Change "rails" to "Rails" throughout
text.gsub!("rails", "Rails")

# Capitalize the word "Rails" throughout
text.gsub!(/\brails\b/, "Rails")

puts "#{text}"
```


References

1. http://ruby-doc.org/
2. https://www.tutorialspoint.com/ruby/ruby_regular_expressions.htm

Thank you for listening!