Regular Expressions

with Examples

Krzysztof Mossakowski
Faculty of Mathematics and Information Science
Warsaw University of Technology, Poland
http://www.mini.pw.edu.pl/~mossakow
**Regular Expressions**

- **Strings used to match and manipulate text**
- **Created using the regular expression language**
  - not a programming language
- **Have a special syntax and instructions**

**Usage:**
- searching
- replacing
Single Characters

- Plain text
  - default: case-sensitive
- . - any character
- \\ . - a dot
Sets of Characters

- \([aAbc]\) - a or A or b or c
- \([A-Z]\) - any uppercase character
- \([A-z]\) - any character between ASCII A and ASCII z
  (also \([\wedge \text{etc.}]\))
- \([0-9]\) - any digit
- \([A-Za-z0-9]\) - any letter or digit
- \([-\wedge0]\) - needs no escaping
- \([\wedge0-9]\) - any nondigit
  (\(^\wedge\) negates all characters in a set)
Metacharacters

- . [ ] + * ? { } ( )

These characters must be escaped when used as normal characters

- `\` - \ (backslash)
- `\[\]` - [] (square brackets)
- `\[[0-9]\]` - square brackets with a digit
Whitespace Characters

- \[\b\] - backspace
- \f - form feed
- \n - line feed (new line)
- \r - carriage return
- \t - tab
- \v - vertical tab
Character Types

- \d  - any digit
- \D  - any nondigit
- \w  \equiv [a-zA-Z0-9_]
- \W  \equiv[^a-zA-Z0-9_]
- \s  - any whitespace
- \S  - any nonwhitespace
- \x# \x##  - character of hexadecimal value
- \0## \0###  - character of octal value
Repeating

- ? - zero or one character
- * - zero or more characters
- + - one or more characters

- https?://[\w./]+ - http or https
  - https? - http or https
  - :// - ://
  - [\w./]+ - alphanumeric, underline, dot or slash - one or more

Krzysztof Mossakowski  (http://www.mini.pw.edu.pl/~mossakow)
Intervals

- \{n\} - exactly \text{n} matches
- \{m,n\} - from \text{m} to \text{n} matches
- \{n,\} - at least \text{n} matches

- \#[0-9A-Fa-f]{6}\} - value of color in HTML
Greedy and Lazy Quantifiers

- **Greedy:**
  - `*`  
  - `+`  
  - `{n,}`

- **Lazy:**
  - `*?`  
  - `+?`  
  - `{n,}?`

There are `<b>two</b>` or `<B>three</B>`.  

- `<[Bb]>.*</[Bb]>`
  1. `<b>two</b>` or `<B>three</B>`

- `<[Bb]>.*?</[Bb]>`
  1. `<b>two</b>`
  2. `<B>three</B>`
Boundaries

- \b - word boundary
- \B - not a word boundary

try cap recap captures this

- cap  : cap recap captures
- \bcap  : cap captures
- cap\b : cap recap
- \bcap\b : cap

a-b - c-d
- \B-\B : a-b - c-d
String Boundaries

- ^ - the start of the string
- $ - the end of the string

In multiline mode: the start and the end of the line

- ^\d : all lines with a digit at the beginning
Subexpressions

- Subexpressions are enclosed between ( and )
- | - OR operator
- Subexpressions can be nested

- hahaaa
  - ha{2} : hahaa
  - (ha){2} : hahaa
- (19|20)\d{2} - 19## or 20##
Backreferences

- reference to a subexpression with number \n
<h1>1st</h1> <h2>2nd</h2> <h1>no</h1>

- <h1>1st</h1> <h2>2nd</h2> <h1>no</h1>
  - <h1>([1-6])</h1>.*?</h1>
  - <h1>1st</h1> <h2>2nd</h2> <h1>no</h1>

- <h1>([1-6])</h1>.*?</h1>
  - <h1>1st</h1> <h2>2nd</h2> <h1>no</h1>
Lookahead & Lookbehind

- ?= - positive lookahead
- ?! - negative lookahead
- ?<= - positive lookbehind
- ?<! - negative lookbehind

name@server.com
- .+@ :name@
- .+(?=@) :name
- @.+ :server.com
- (?<=@).* :server.com
.NET Classes

- System.Text.RegularExpressions.Regex
  - regular expression string and a set of options
- System.Text.RegularExpressions.Match
  - occurrences of the regular expression pattern
  - access to text segments via group names or numbers
- System.Text.RegularExpressions.Group
  - text segments from the input text that were mapped to group name or numbers
- System.Text.RegularExpressions.RegexOptions
  - enumeration values for options
Methods of Regex class

- **IsMatch()** - checks to see whether a match is found in a specified string.
- **Match()** - searches for a single match, which is returned as a **Match** object.
- **Matches()** - searches for all matches, which are returned as a **MatchCollection** object.
- **Replace()** - performs a replace operation on a specified string.
- **Split()** - splits a string into an array of strings.
RegexOptions

- **Compiled** – faster execution, slower startup
- **IgnoreCase** – for case-insensitive matching
- **Multiline** - ^ and $ work for lines, not whole text
- **Singleline** - changes the meaning of the dot (.) so it matches every character (instead of every character except\n)
Examples
HREFs in HTML

Listing all links from HTML file

```csharp
void DumpHrefs(String input)
{
    Regex r;
    Match m;

    r = new Regex(
        @"href\s*=[\s]*(?="\([^"\]*\)"|(?<1>|\S+))",
        RegexOptions.IgnoreCase | RegexOptions.Compiled);
    for (m = r.Match(input); m.Success; m = m.NextMatch())
    {
        Console.WriteLine(
            "Found href " + m.Groups[1] + " at " + m.Groups[1].Index);
    }
}
```
Explanation

@"href\s*=\s*({=""\s*(<[^>]+>|[^> \t]*)\s*})\s*""\s*([^> \s]+)\s*"

lookahead
(skip quotation mark)

any number of whitespaces

any number of any characters except for quotation mark

plain text
(quote mark)

one or more nonwhitespaces

named group

the first possibility

the second possibility

subexpression
Date Formats

- Converting dates from mm/dd/yy to dd-mm-yy

```csharp
string MDYToDMY(string input)
{
    return Regex.Replace(input,
"\\b(?<month>\\d{1,2})/(?<day>\\d{1,2})/(?<year>\\d{2,4})\\b",
"${day}-${month}-${year}");
}
```
Explanation

@"\b(?<month>\d{1,2})/(?<day>\d{1,2})/(?<year>\d{2,4})\b",

digested
	named groups

text

word boundary
Invalid Characters

- Stripping invalid characters from a string

```csharp
String CleanInput(string strIn)
{
    // Replace invalid characters with empty strings.
    return Regex.Replace(strIn, @"[^\w\.\@-]", "");
}
```
Tag Parser

- Lists all tags from XML file with indents

```csharp
public static void showTags(String text, String indent)
{
    Regex tagRegex = new Regex(@"<(?<tag>\w+)>(?<content>(. |\n)*?)</\k<tag>>");
    Match tagMatch = tagRegex.Match(text);
    for ( ; tagMatch.Success; tagMatch = tagMatch.NextMatch())
    {
        System.Console.WriteLine(
            indent+"<"+tagMatch.Groups["tag"]+">");
        showTags(""+tagMatch.Groups["content"],indent+"    ");
        System.Console.WriteLine(
            indent+"</"+tagMatch.Groups["tag"]+">");
    }
}
```
Explanation

@"<(?<tag>\w+)>(?<content>\.( |\n)*?)</\k<tag>>"
IP Address

(((\d{1,2}) | (1\d{2}) | (2[0-4]\d) | (25[0-5])) \.) {3} (((\d{1,2}) | (1\d{2}) | (2[0-4]\d) | (25[0-5]))

for each part:
- any one- or two-digit number
- any three-digit number beginning with 1
- any three-digit number beginning with 2
  if the second digit is 0 through 4
- any three-digit number beginning with 25
  if the third digit is 0 through 5
Complete URL

- http://name:pass@www.server.com/path/to/file/index.html?param1=value1&param2=value2