# C# Regular Expressions

**Hans-Wolfgang Loidl**
<H.W.Loidl@hw.ac.uk>

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh


**Semester 1 — 2018/19**




# Regular Expressions in C#

- ***Regular expressions*** **provide a powerful, efficient and flexible text processing technique.**
- **They form the basis of text and data processing tools.**
- **There are standardised versions of regular expressions (eg. POSIX) as well as extensions for specific packages and languages (eg. Perl).**

**See**
**this single-page quick reference on C# regular expressions.**




# Using Regular Expressions

- **Integrated in many tools (vi, grep, etc) and languages(C#, Perl, etc).**
- **Provides a lot of features:**
  - **Match upper and lower case.**
  - **Either pattern or string matching.**
  - **Quantify character repeats.**
  - **Match classes of characters.**
  - **Match any character.**
  - **Expressions can be combined.**
  - **You can match anything using regular expressions.**
- **Syntax is simple.**




# Regular Expression Language

- **Regular expressions are constructed using two types of characters:**
  - **Special characters or meta characters**
  - **Literal or normal text.**
- **Can think of regular expressions as a language:**
  - **Grammar: meta characters.**
  - **Words: literal text.**

# Single Characters

- `.` The dot matches *any* single character. E.g.
  - `ab.` matches `aba`, `abb`, `abc`, etc.
- `[ ]` A bracket expression matches a *single character* contained within the bracket. E.g.
  - `[abc]` matches `a`, `b` or `c`
  - `[a-z]` specifies a range which matches any lowercase letter from `a` to `z`.
  - `[abcx-z]` matches `a`, `b`, `c`, `x`, `y` and `z`.



# Control Characters

- `\t` horizontal tab (`\u0009`)
- `\v` vertical tab (`\u000b`)
- `\b` backspace (`\u0008`)
- `\e` escape (`\u0018`)
- `\r` carriage return (`\u000d`)
- `\f` form feed (`\u000c`)
- `\n` new line (`\u000a`)



# Meta Characters

- `[^ ]` negation of `[ ]` i.e. matches a single character *not contained* in bracket. E.g.
  - `[^abc]` matches any character other than `a`, `b` or `c`.
- `^` matches the starting position of a string.
- `$` matches the ending position of a string.
- `\b` matches a word boundary
- `?` matches the previous element *zero* or once.
- `*` matches the previous element *zero* or more times. E.g.
  - `abc*d` matches `abd`, `abcd`, `abccd`, etc.
- `+` matches the previous element *one* or more times. E.g.
  - `abc*d` matches `abcd`, `abccd`, etc.



# Character classes

- `\w` word character
- `\W` non-word character
- `\d` decimal digit
- `\D` non-decimal digit
- `\s` white-space
- `\S` non-white-space

# POSIX Regular Expressions

These are regular expressions standardised by a POSIX document:

- `[:alnum:]` matches alpha-numerical characters
- `[:alpha:]` matches alphabetical characters
- `[:digit:]` matches numerals
- `[:upper:]` matches upper case characters
- `[:lower:]` matches lower case characters



# Examples

Example regular expressions:

- Find the word 'whatever' in some text: `\bwhatever\b`
- Find monetary values such as '34 GBP' in a text: `\d+\s*GBP`
- Find either `Console.Write` or `Console.WriteLine` in program code[1]: `Console\.Write(Line)?`

[1] Source https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.match?view=netframework-4.7.2



# Groups in Regular Expressions

A regular expression can contain groups, which can be referred to by index or name:

- `(exp)` indexed group, captured as index `\1`
- `\1` reference to first indexed group in a pattern
- `(?<word>)` named group, captured with the name `word`
- `\k<word>` reference to a named group, captured with the name `word`

Examples:

- Find repeated words in a string (indexed groups): `(\w+)\s+\1\b`
- Find repeated words in a string (named groups): `(?<word>\w+)\s+\k<word>\b`



# Example: Find repeated words

```
1  // Define a regular expression for repeated words.
2  Regex rx = new Regex(@"\b(?<word>\w+)\s+(\k<word>)\b",
3    RegexOptions.Compiled | RegexOptions.IgnoreCase);
4
5  // Define a test string.
6  string text = "The␣the␣quick␣brown␣fox␣␣fox␣jumps␣over
       ␣the␣lazy␣dog␣dog.";
7
8  // Find matches.
9  MatchCollection matches = rx.Matches(text);
10 // Report on each match.
11 foreach (Match match in matches) {
12   GroupCollection groups = match.Groups;
13   Console.WriteLine("'{0}'␣repeated␣at␣positions␣{1}␣
         and␣{2}",
14                     groups["word"].Value, groups[0].
                           Index, groups[1].Index);   }
```

[1] regexp1.cs;
Source: https://docs.microsoft.com/en-

# Examples

**Process a file/string with information about Scottish cities (from Wikipedia):**

```
Rank  Locality  Population  Status  Council area
1  Glasgow  599,650  City  Glasgow City
2  Edinburgh  464,990  City  City of Edinburgh
3  Aberdeen  196,670  City  Aberdeen City
4  Dundee  147,710  City  Dundee City
5  Paisley  76,220  Town  Renfrewshire
6  East Kilbride  74,740  Town  South Lanarkshire
```

- **Finding all city names:** `^[0-9]+\s+([A-Za-z]*).*$`



# Example Program

**Finding a single match in a string, using indexed groups:**

```
1  // Define a regular expression
2  Regex rx1 = new Regex(@"^[0-9]+\s+([A-Za-z]*).*$",
3    RegexOptions.Compiled);
4  // test string to match against
5  string text = @"2␣␣␣␣␣␣␣Edinburgh␣␣␣␣␣␣␣464,990␣
       ␣␣␣␣␣␣␣␣City␣␣␣␣City␣of␣Edinburgh";
6  // Find single match
7  Match match1 = rx1.Match(text);
8
9  // check and print
10 if (match1.Success) {
11   Console.WriteLine("First␣group␣'{0}'␣in␣matched␣sub-
         string␣'{1}'␣found␣in␣single␣line:\n{2}",
12                     match1.Groups[1], match1.Value,
                           text);
13 }
```

[1] regexp4.cs



# Example Program

**Finding a several matches in a string (named groups):**

```
1  // Define a regular expression for repeated words.
2  Regex rx = new Regex(@"\b(?<word>\w+)\s+(\k<word>)\b",
3    RegexOptions.Compiled | RegexOptions.IgnoreCase);
4
5  // Define a test string.
6  string text = "The␣the␣quick␣brown␣fox␣␣fox␣jumps␣over
       ␣the␣lazy␣dog␣dog.";
7
8  // Find matches.
9  MatchCollection matches = rx.Matches(text);
10
11 // Report on each match.
12 foreach (Match match in matches) {
13     GroupCollection groups = match.Groups;
14     Console.WriteLine("'{0}'␣repeated␣at␣positions␣{1}
           ␣and␣{2}",
15                       groups["word"].Value, groups[0].
                             Index, groups[1].Index);
```

[1] regexp1.cs



# Resources

**External resources:**

- **MSDN web page on C# regular expressions**
- **MSDN C# regular expressions quick reference**
- **See this single-page quick reference on C# regular expressions.**

**Sample code:**

- **regexp1.cs**
- **regexp4.cs**