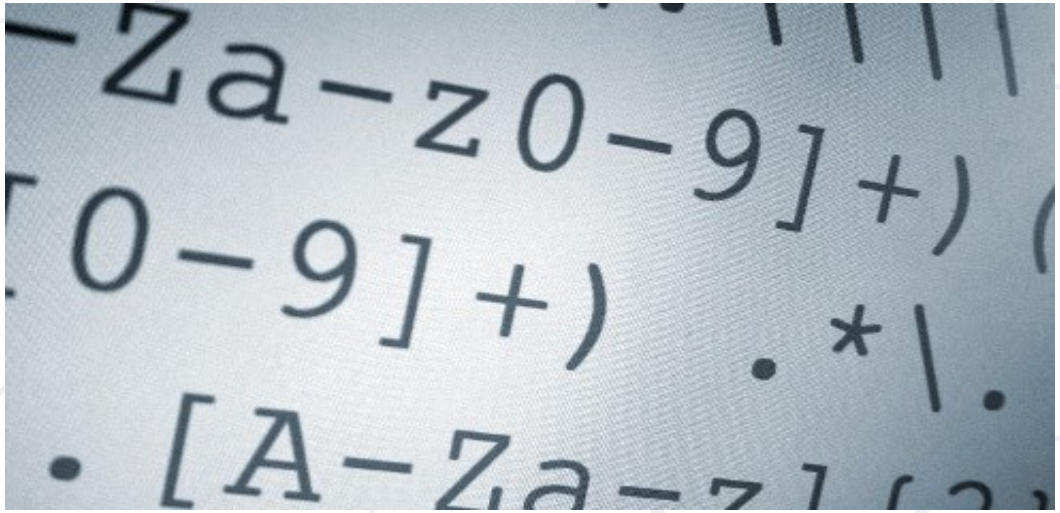
Regular Expressions (RegEx)

Regular Expressions Language Syntax



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

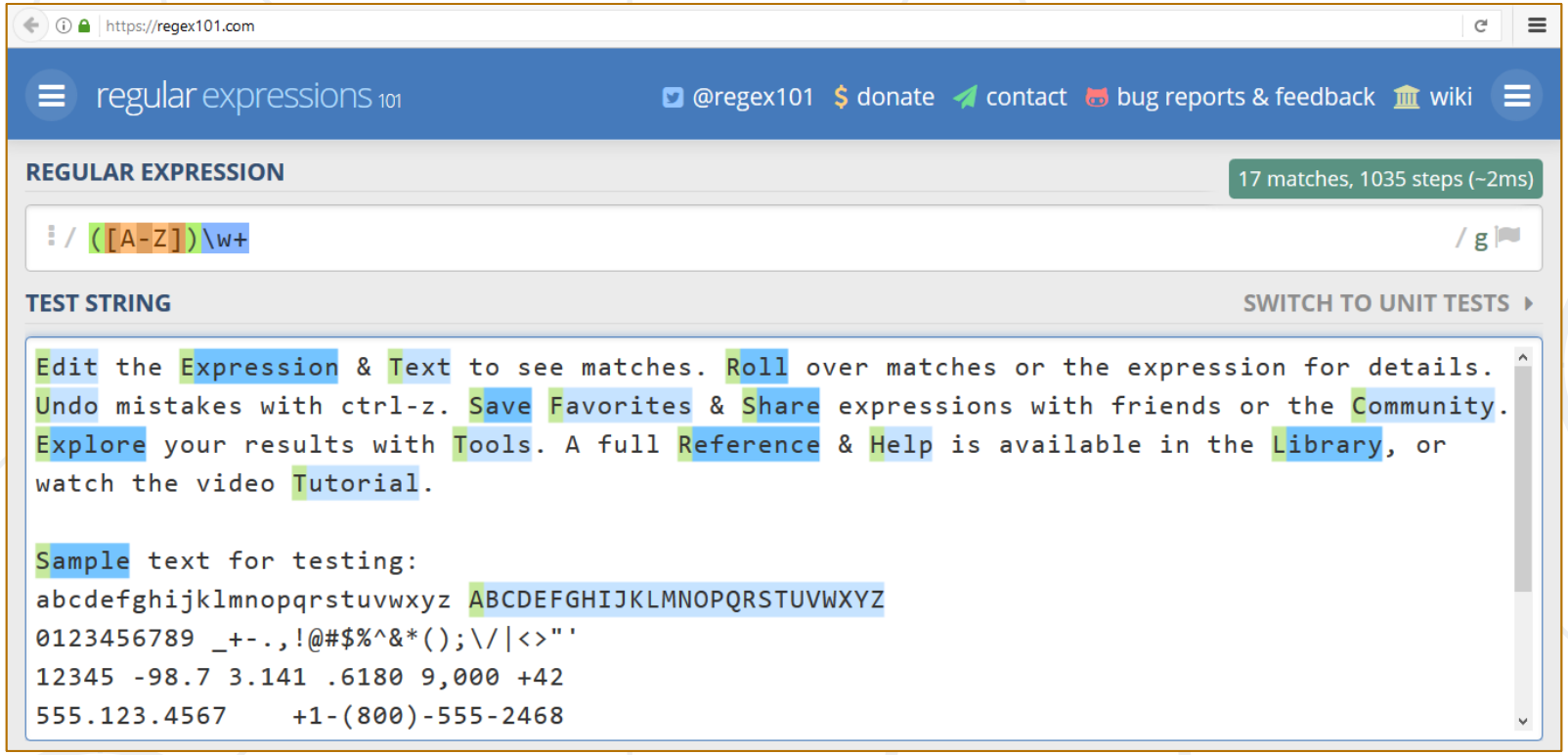
Table of Contents

[A-Z]

Regular Expressions

What Are Regular Expressions?

- **Regular expressions** (regex)
  - Match text by **pattern**
- Patterns are defined by special syntax, e.g.
  - `[0-9]+` matches non-empty sequence of digits
  - `[A-Z][a-z]*` matches a capital + small letters
- Play with regex live at: regexr.com, regex101.com


https://regex101.com
regular expressions 101
@regex101
donate
contact
bug reports & feedback
wiki
REGULAR EXPRESSION
17 matches, 1035 steps (~2ms)
/ ([A-Z])\w+
/ g
TEST STRING
SWITCH TO UNIT TESTS
Edit the Expression & Text to see matches. Roll over matches or the expression for details.
Undo mistakes with ctrl-z. Save Favorites & Share expressions with friends or the Community.
Explore your results with Tools. A full Reference & Help is available in the Library, or
watch the video Tutorial.
Sample text for testing:
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 _+-.,!@#$%^&*();\/|<>"'
12345 -98.7 3.141 .6180 9,000 +42
555.123.4567 +1-(800)-555-2468

Live Demo

Regular Expression Pattern – Example

- **Regular expressions** (regex) describe a **search pattern**
- Used to **find** / **extract** / **replace** / **split** data from text by pattern

```
[A-Z][a-z]+ [A-Z][a-z]+
```

```
John Smith
```

```
Linda Davis
```

```
Contact: Alex Scott
```

Character Classes: Ranges

- `[nvj]` matches any character that is either **n**, **v** or **j**

```
node.js v0.12.2
```

- `[^abc]` – matches any character that is **not** **a**, **b** or **c**

```
Abraham
```

- `[0-9]` – character range: matches any digit from **0** to **9**

```
John is 8 years old.
```

Predefined Classes

- **\w** – matches any **word character** (a-z, A-Z, 0-9, _)
- **\W** – matches any **non-word character** (the opposite of **\w**)
- **\s** – matches any **white-space** character
- **\S** – matches any **non-white-space** character (opposite of **\s**)
- **\b** – matches the **border** between space and non-space
- **\d** – matches any **decimal digit** (0-9)
- **\D** – matches any **non-decimal character** (the opposite of **\d**)

(\w+)

Quantifiers

Quantifiers

- * - matches the previous element **zero or more times**

```
\+\d*
```

```
+359885976002 a+b
```

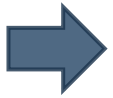
- + - matches the previous element **one or more times**

```
\+\d+
```

```
+359885976002 a+b
```

- ? - matches the previous element **zero or one time**

```
\+\d?
```



```
+359885976002 a+b
```

- {3} - matches the previous element **exactly 3 times**

```
\+\d{3}
```

```
+359885976002 a+b
```

Grouping Constructs

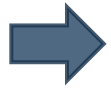
- `(subexpression)` - captures the matched subexpression as numbered group

```
\d{2}-(\w{3})-\d{4}
```

```
22-Jan-2015
```

- `(?:subexpression)` - defines a non-capturing group

```
^(?:Hi|hello),\s*(\w+)$
```



```
Hi, Peter
```

- `(?<name>subexpression)` - defines a named capturing group

```
(?<day>\d{2})-(?<month>\w{3})-
(?<year>\d{4})
```

```
22-Jan-2015
```

Problem: Match All Words

- Write a regular expression in www.regex101.com that extracts all word char sequences from given text

```
_ (Underscores) are also word characters!
```

```
_|Underscores|are|also|word|characters
```

Solution: Match All Words

```
string pattern = @"\w+";
```

\w+ matches word characters one or more times

Problem: Match Dates

- Write a regular expression that extracts **dates** from text
  - Valid date format: **dd-MMM-yyyy**
  - Examples: **12-Jun-1999**, **3-Nov-1999**

```
I was born on 30-Dec-1994. My father was
born on the 9-Jul-1955. 01-July-2000 is
not a valid date.
```

Solution: Match Dates

\d? matches a digit zero or one time

\d matches a digit

\d{4} matches exactly 4 digits

```
string pattern = @"\d?\d-[A-Z][a-z]{2}-\d{4}";
```

[A-Z] matches a capital letter

[a-z]{2} matches exactly 2 lower case letters

```
I was born on 30-Dec-1994 in Sofia.
```

Problem: Email Validation

- Write a regular expression that performs simple **email validation**
  - An email consists of: **username @ domain name**
  - **Usernames** are **alphanumeric**
  - **Domain names** consist of **two strings**, separated by a **period**
  - **Domain names** may contain only **English letters**

Valid: `valid123@email.bg`

Valid: `hi@mail.abv.bg`

Invalid: `invalid*name@emai1.bg`

Invalid: `pesho@abv.`

Solution: Email Validation

^ asserts position at start of a line

\w+ matches a sequence of word chars

$ asserts position at end of a line

```
string pattern = @"^\w+@(\w+\.)+\w+$";
```

@ matches the character "@"

\. matches exactly "."

(\w+\.)+ matches several words + "."

```
hi@mail.abv.bg
```

\1

Backreferences

Backreferences Match Previous Groups

- \number - matches the value of a numbered capture group

```
<(\w+)[^>]*>.*?<\/\1>
```

```
<b>Regular Expressions</b> are cool!
<p>I am a paragraph</p> … some text after
Hello, <div>I am a<code>DIV</code></div>!
<span>Hello, I am Span</span>
<a href="https://softuni.bg/">SoftUni</a>
```

```
17          string pattern = @"^\w+@\w+\.\w+$";
18
19          List<string> emails = new List<string>()
20          {
21              "dow_jones@gmail.com",
22              "spam@nakov",
23              "JonSkeet69@1337.org",
24              "ayy lmao@abv.bg"
25          };
26
27          Regex regex = new Regex(pattern);
28          foreach (var email in emails)
29          {
30              Console.WriteLine(regex.IsMatch(email));
31          }
```

Using .NET Built-In Regex Classes

Regex in C#

- C# supports a built-in regular expression class: **Regex**
  - Located in **System.Text.RegularExpressions** namespace

```
using System.Text.RegularExpressions;

static void Main()
{
  string pattern = @"A\w+";
  Regex regex = new Regex(pattern);
}
```

Validating String by Pattern

- **IsMatch(string text)**
  - Determines whether the text matches given pattern

```
string text = "Today is 2015-05-11";
string pattern = @"\d{4}-\d{2}-\d{2}";

Regex regex = new Regex(pattern);
bool containsValidDate = regex.IsMatch(text);

Console.WriteLine(containsValidDate); // True
```

Checking for a Single Match

- **Match(string text)**
  - Returns the first match of given pattern

```
string text = "Nakov: 123";
string pattern = @"([A-Z][a-z]+): (\d+)";
Regex regex = new Regex(pattern);
Match match = regex.Match(text);

Console.WriteLine(match.Groups.Count); // 3
Console.WriteLine("Matched text: \"{0}\"", match.Groups[0]);
Console.WriteLine("Name: {0}", match.Groups[1]); // Nakov
Console.WriteLine("Number: {0}", match.Groups[2]); // 123
```

Checking for Matches

- **Matches(string text)** - returns a collection of matches

```
string text = "Nakov: 123, Branson: 456";
string pattern = @"([A-Z][a-z]+): (\d+)";

Regex regex = new Regex(pattern);

MatchCollection matches = regex.Matches(text);
Console.WriteLine("Found {0} matches", matches.Count);

foreach (Match match in matches)
  Console.WriteLine("Name: {0}", match.Groups[1]);

// Found 2 matches
// Name: Nakov
// Name: Branson
```

Replacing with Regex

- **Replace(string text, string replacement)** - replaces all strings that match the pattern with the provided replacement

```
string text = "Nakov: 123, Branson: 456";
string pattern = @"\d{3}";
string replacement = "999";

Regex regex = new Regex(pattern);
string result = regex.Replace(text, replacement);

Console.WriteLine(result);
// Nakov: 999, Branson: 999
```

Splitting with Regex

- **Split(string text)** - splits the text by the pattern
  - Returns string[]

```
string text = "1   2 3      4";
string pattern = @"\s+";

string[] results = Regex.Split(text, pattern);
Console.WriteLine(string.Join(", ", results));
// 1, 2, 3, 4
```

Problem: Match Full Name

- You are given a **list of names**
  - Match all **full names** (two words, starting with capital letter)

```
Ivan Ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Test
Testov, Ivan   Ivanov
```

```
Ivan Ivanov
Test Testov
```

Solution: Match Full Names

```
string listOfNames = Console.ReadLine();

string pattern @"\b[A-Z][a-z]+ [A-Z][a-z]+";
Regex regex = new Regex(pattern);

MatchCollection validNames = regex.Matches(input);

foreach (Match name in validNames)
{
  Console.Write($"{name.Value}" + "\n");
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3178#21

Problem: Match Dates

- You are given a string
  - Match all dates in the format "**dd{separator}MMM{separator}yyyy**" and print them space-separated

```
13/Jul/1928, 01/Jan-1951
```

```
Day: 13, Month: Jul, Year: 1928
```

Solution: Match Dates

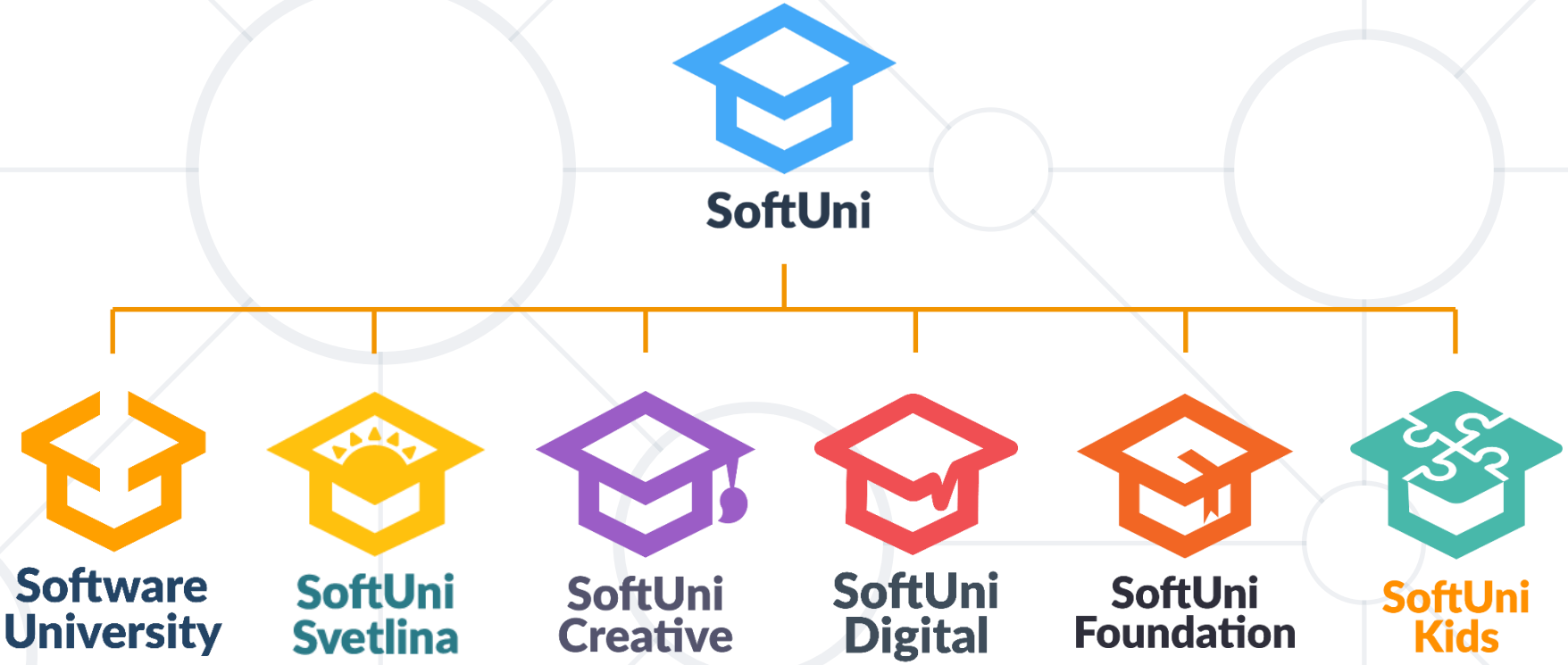
```
string input = Console.ReadLine();

string pattern = @"\b(?<day>\d{2})(\.|-|\/)
(?<month>[A-Z][a-z]{2})\1(?<year>\d{4})\b";

MatchCollection matches = Regex.Matches(input, pattern);

foreach (Match date in matches)
    Console.WriteLine($"Day: {date.Groups["day"].Value},
    Month: {date.Groups["month"].Value}, Year:
    {date.Groups["year"].Value}");
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3178#23

Summary

- **Regular expressions** describe **patterns** for searching through text
- Define **special characters**, **operators** and **constructs** for building complex pattern
- Can utilize **character classes**, **groups**, **quantifiers**, etc.
- In C# use the **Regex** class

Questions?



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg