Budapest University of Technology and Economics
Department of Automation and Applied Iformatics

informatics II Labs

# Regular Expression

Khalid Kahloot, Dmitriy Dunaev
{kkahloots, dunaev}@aut.bme.hu

BME AUT
Department of
Automation and
Applied Informatics

# What is Regular Expression?

- A pattern that is matched against a subject string from left to right.
- Example: Username
  - allow the username to contain letters, numbers, underscores and hyphens.
  - limit the number of characters in username

Start of the line

3 to 15 characters long

```
^[a-z0-9_-]{3,15}$
```

End of the line

Letters, numbers, underscores, hypens

# Regex Language Elements

- **Metacharacters**

  - The constructs within regular expressions that have special meaning are referred to as metacharacters

  - Characters other than . $ ^ { [ ( | ) ] } * + ? \ match themselves.

- **Character Classes**

- Character classes are a mini-language within regular expressions, defined by the enclosing hard braces **[ ]**.

  **Examples: [a-z A-Z 0-9],**

# Regular Expression Quick Guide

| | |
|---|---|
| ^ | Matches the beginning of a line |
| $ | Matches the end of the line |
| . | Matches any character |
| \s | Matches whitespace |
| \S | Matches any non-whitespace character |
| * | Repeats a character zero or more times |
| *? | Repeats a character zero or more times (non-greedy) |
| + | Repeats a chracter one or more times |
| +? | Repeats a character one or more times (non-greedy) |
| [aeiou] | Matches a single character in the listed set |
| [^XYZ] | Matches a single character not in the listed set |
| [a-z0-9] | The set of characters can include a range |
| ( | Indicates where string extraction is to start |
| ) | Indicates where string extraction is to end |

# A Single Character of: a, b OR c

`[abc]` Matches either an a, b or c character

```
/[abc]+/g
a bb ccc
```

# A Character Except: a, b OR c

`[^abc]` Matches any character except for an a, b or c

```
/[^abc]+/g
Anything but abc.
```

# A Character in Range: a-z

`[a-z]` Matches any characters between a and z, including a and z.

```
/[a-z]+/g

Only a-z
```

# A Character in Range: a-z OR A-Z

`[a-zA-Z]` Matches any characters between a-z or A-Z. You can combine as much as you please.

```
/[a-zA-Z]+/g
abc123DEF
```

# ANY single character

. Matches any character other than newline (or including newline with the /s flag)

```
/.+/

a b c
```

# ANY Whitespace character

`\s` Matches any space, tab or newline character.

```
/\s/g

any whitespace character
```

# ANY non-Whitespace character

`\S` Matches anything other than a space, tab or newline.

```
/\S+/

any non-whitespace
```

# ANY digit

`\d` Matches any decimal digit. Equivalent to [0-9].

```
/\d/g
one: 1, two: 2
```

# ANY non-digit

`\D` Matches anything other than a decimal digit.

```
/\D+/
one: 1, two: 2
```

# ANY Word Character

`\w` Matches any letter, digit or underscore. Equivalent to [a-zA-Z0-9_].

```
/\w+/g
```

any word character

# Capture everything enclosed

`(...)` Parts of the regex enclosed in parentheses may be referred to later in the expression or extracted from the results of a successful match.

```
/(he)+/g

heheh he heh
```

# Match either a or b

(a|b) Matches the a or the b part of the subexpression.

```
/(a|b)/g
beach
```

# ZERO OR one of ‘a’

a? Matches an `a` character or nothing.

```
/ba?/g
ba b a
```

# ZERO OR more of ‘a’

a* Matches zero or more consecutive `a` characters.

```
/ba*/g
a ba baa aaa ba b
```

# EXACTLY 3 of ‘a’

`a{3}` Matches exactly 3 consecutive `a` characters.

```
/a{3}/g

a  aa  aaa  aaaa
```

# 3 OR more of ‘a’

`a{3,}` Matches at least 3 consecutive `a` characters.

```
/a{3,}/
a aa aaa aaaa aaaaaa
```

# BETWEEN 3 AND 6 of ‘a’

`a{3,6}` Matches between 3 and 6 (inclusive) consecutive `a` characters.

```
/a{3,6}/

a  aa  aaa  aaaa  aaaaaaaaaa
```

# Start of String

^ Matches the start of a string without consuming any characters. If multiline mode is used, this will also match immediately after a newline character.

```
/^\w+/

start of string
```

# End of String

$ Matches the end of a string without consuming any characters. If multiline mode is used, this will also match immediately before a newline character.

```
/\w+$/

end of string
```

# A Word Boundary

`\b` Matches, without consuming any characters, immediately between a character matched by \w and a character not matched by \w (in either order). It cannot be used to separate non words from words.

```
/d\b/g

word boundaries are odd
```

# Examples

| Pattern | Description |
| --- | --- |
| • ^\d{5}$ | 5 numeric digits, such as a US ZIP code. |
| • ^\d{3}-\d{2}-\d{4}$ | Validates the format such as 111-11-1111 (Social Security Number) |
| • ^[01]?[- .]? (\([2-9]\d{2}\) \| [2-9]\d{2}) [- .]? \d{3}[- .]? \d{4}$ | Validates a U.S. phone number. |

# Regular Expression API

- Regular Expression classes found in the **System.Text.RegularExpressions** namespace.
- the main classes we'll want to use are **Regex**, **Match**, and **MatchCollection.**
- **Regex:** Methods and their description

**- IsMatch -** Returns true if the regex finds a match in the input string.

- **Match -** Returns a Match object if a match is found in the input string.

- **Matches -** Returns a MatchCollection object containing any and all matches found in the input string.

- **Replace** - Replaces matches in the input string with a given replacement string.

# Cont…

- **Capture** :

  - Represents the results from a single subexpression capture.

  - Capture represents one substring for a single successful capture

- **CaptureCollection** :

  - Represents a sequence of capture substrings.

  - CaptureCollection returns the set of captures done by a single capturing group

- **Group**:

  - Group represents the results from a single capturing group.

  - A capturing group can capture zero, one, or more strings in a single match because of quantifiers, so Group supplies a collection of Capture objects.

- **GroupCollection** :

  - Represents a collection of captured groups.

  - GroupCollection returns the set of captured groups in a single match.

# 1. Basic Matchers

- Pattern : "the"
  - the letter t, followed by the letter h, followed by the letter e.
-

# 1. Basic Matchers

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string pattern = @"the";
        string input = @"The fat cat sat on the mat.";

        foreach (Match m in Regex.Matches(input, pattern))
        {
            Console.WriteLine("'{0}' found at index {1}.", m.Value, m.Index);
        }
    }
}
```

# 2. Meta Characters

- **2.1 Full stop**

- Full stop . is the simplest example of meta character.

- The meta character . matches any single character. It will not match return or newline characters.

- For example, the regular expression .ar means: any character, followed by the letter a, followed by the letter r.

# 2.1 **Full stop**

Write a regular expression for the follow pattern:

any character, followed by the letter a,
followed by the letter r.

```
The car parked in the garage.
```

# 2.2 Character set

Write a regular expression for the follow pattern:

a lowercase character a, followed by letter r, followed by a period . character.

```
A garage is a good place to park a car.
```

# 2.2.1 Negated character set

Write a regular expression for the follow pattern:

any character except c, followed by the character a, followed by the letter r.

```
The car parked in the garage.
```

# 2.3 Repetitions

- Following meta characters +, * or ? are used to specify how many times a subpattern can occur.

- These meta characters act differently in different situations.

# 2.3.1 The Star

Write a regular expression for the follow pattern:

- any number of lowercase letters in a row.

```
The car parked in the garage #21.
```

-zero or more spaces, followed by lowercase character c,
followed by lowercase character a,
followed by lowercase character t,
followed by zero or more spaces.

```
The fat cat sat on the concatenation.
```

# 2.3.2 The Plus

Write a regular expression for the follow pattern:

lowercase letter c, followed by at least one character, followed by the lowercase character t.
It needs to be clarified that t is the last t in the sentence.

```
The fat cat sat on the mat.
```

# 2.3.3 The Question Mark

Write a regular expression for the follow pattern:

Optional the uppercase letter T,
followed by the lowercase character h,
followed by the lowercase character e.

```
The car is parked in the garage.
```

```
The car is parked in the garage.
```

# 2.4 Braces

Write a regular expression for the follow pattern:

Match at least 2 digits but not more than 3
( characters in the range of 0 to 9).

```
The number was 9.9997 but we rounded it off to 10.0.
```

# 2.4 Braces - cont

Write a regular expression for the follow pattern:

Match 2 or more digits. If we also remove the comma
the regular expression [0-9]{3} means: Match exactly 3 digits.

```
The number was 9.9997 but we rounded it off to 10.0.
```

```
The number was 9.9997 but we rounded it off to 10.0.
```

# 2.5 Character Group

Write a regular expression for the follow pattern:

lowercase character c, g or p,
followed by character a, followed by character r.

```
The car is parked in the garage.
```

# 2.8 matching

```
// Example 1 matches words that start with 'S'
str = "A Thousand Splendid Suns";

// Example 2 matches words that start with 'm' and ends with 'e'
str = "make maze and manage to measure it";

// Example 3 Matching and lists the matched groups, captures,
//and capture positions
 str = "One car red car blue car";

// Example 4 non-Greedy and Greedy Backtracking
 str = "abcd 88abcd";
 part = "ab";
```