# .NET Framework 4.7 and C# 8.0

## Lesson 06 : Evaluating Regular Expressions

Capgemini

# Evaluating Regular Expressions

- Introduction
- Forming Regular Expression
- Regex Class
- Regex Syntax
- IsMatch
- Matches
- Replace
- Split

# Introduction

- Regular expressions are a pattern matching standard for string parsing and replacement and is a way for a computer user to express how a computer program should look for a specified pattern in text and then what the program is to do when each pattern match is found.

- The regular expression engine in .NET is a powerful, full-featured tool that processes text based on pattern matches rather than on comparing and matching literal text.

- In most cases, it performs pattern matching rapidly and efficiently. However, in some cases, the regular expression engine can appear to be very slow.

# The Regex Class

- C# supports regular expressions through the classes in the **System.Text.RegularExpressions** namespace in the standard .NET framework.

# Forming Regular Expressions

- There are three important parts to a regular expression.
  - Anchors are used to specify the position of the pattern in relation to a line of text.
  - Character Sets match one or more characters in a single position.
  - Modifiers specify how many times the previous character set is repeated.

- Example:
  - **^#***
    - **^** - Indicates beginning of line
    - **#** - Character set that matches single character
    - ***** - Modifier that specifies the how many time the previous character set will repeat

# Forming Regular Expressions

- Anchor Characters (^ and $):

| Pattern | Matches |
|---|---|
| ^A | "A" at the beginning of a line |
| A$ | "A" at the end of a line |
| A^ | "A^" anywhere on a line |
| $A | "$A" anywhere on a line |
| ^^ | "^" at the beginning of a line |
| $$ | "$" at the end of a line |

# Forming Regular Expressions

- Character Sets:

| Regular Expression | Matches |
|---|---|
| [] | The characters "[]" |
| [0] | The character "0" |
| [0-9] | Any number |
| [^0-9] | Any character other than a number |
| [-0-9] | Any number or a "-" |
| [0-9-] | Any number or a "-" |
| [^-0-9] | Any character except a number or a "-" |
| []0-9] | Any number or a "]" |
| [0-9]] | Any number followed by a "]" |
| [0-9-z] | Any number, or any character between "9" and "z" |
| [0-9\-a\]] | Any number, or a "-", a "a", or a "]" |

# Some Samples

| Quantifier | Description | Regex | Matches |
|---|---|---|---|
| * | Matches the preceding character zero or more times. | a*b | b, ab,aab,aaab…etc |
| + | Matches the preceding character 1 or more times | a+b | Ab,aab,aaab… etc |
| ? | Matches the preceding char zero or one time | a?b | b, ab |
| ^ | It is used to match the beginning of a string. | ^ Capgemini | Capgemini holds the strength of more than 1 lakh employees in India. |
| $ | It is used to match the end of a string. | $ Capgemini | I work with Capgemini |
| . | Matches any character only once. | C.P | CAP, CEP, COP |

# Some Samples

| Quantifier | Description |
| --- | --- |
| \s* | Match zero or more white-space characters. |
| \s? | Match zero or one white-space character. |
| \d? | Match zero or more decimal digits. |

# Examples

- Literals and Special Characters:

  **REGEX: TX**

  **INPUT:** TX **MATCH:** true

  **INPUT:** AZ **MATCH:** false

- Character Range:

  **REGEX:** [013][FXB]

  **INPUT:** 1X **MATCH:** true

  **INPUT:** 1Z **MATCH:** false

# Examples

Character Range: (Contd)

**REGEX:** [A-Za-z0-9][0-9]

**INPUT:** i5 **MATCH:** true

**INPUT:** 1X **MATCH:** false

**REGEX:** [^AEIOU]

**INPUT:** X **MATCH:** true

**INPUT:** E **MATCH:** false

# Examples

- Quantifiers:

**REGEX:** [A-Z][A-Z][A-Z]

**INPUT:** YCA **MATCH:** true

**REGEX:** [A-Z]{3}

**INPUT:** YCA **MATCH:** true

**REGEX:** [0-9]{3}-[0-9]{3}-[0-9]{4}

**INPUT:** 470-127-7501 **MATCH:** true

**INPUT:** 75663-2372 **MATCH:** false

# Examples

**REGEX:** [A-Za-z0-9]{2,}

**INPUT:** YZ1 **MATCH:** true

**INPUT:** YZSDjhfhSBH2342SDFSDFsdfw123412 **MATCH:** true

**REGEX:** [0-3]+[XYZ]*

**INPUT:** 34 **MATCH:** true

**INPUT:** 34YYXZZ **MATCH:** true

# Examples

- Alternation:

**REGEX:** [0-9]{3}(35|75)

**INPUT:** 75035 **MATCH:** true

**INPUT:** 75062 **MATCH:** false

# IsMatch

- public bool IsMatch(string input):
  - Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string.

- public bool IsMatch(string input, int startat)
  - Indicates whether the regular expression specified in the Regex constructor finds a match in the specified input string, beginning at the specified starting position in the string.

- public static bool IsMatch(string input, string pattern)
  - Indicates whether the specified regular expression finds a match in the specified input string

# Match

- public MatchCollection Matches(string input)
  - Searches the specified input string for all occurrences of a regular expression.

# Matches

- public MatchCollection Matches(string input)
  - Searches the specified input string for all occurrences of a regular expression.

# Replace

- public string Replace(string input, string replacement)
  - In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string.

# Split

- public string[] Split(string input)
  - Splits an input string into an array of substrings at the positions defined by a regular expression pattern specified in the Regex constructor.