Certainly! In C#, the **System.Text.RegularExpressions** namespace provides classes for working with regular expressions. Regular expressions (regex) are powerful tools for pattern matching and manipulation of strings. Here's an overview of how the regex library functions, along with examples of patterns, separators, anchors, and more.

**Basic Functions:**

1. **Regex Class:** The **Regex** class is the heart of the regex library in C#. You use this class to define a regular expression pattern and then perform various operations like matching, replacing, and splitting strings.

csharpCopy code

using System.Text.RegularExpressions;

1. **Pattern Matching:** Regular expressions consist of a combination of literal characters and special symbols that form a search pattern. For example:
   * **.**: Matches any single character except for a newline.
   * **\***: Matches zero or more occurrences of the preceding element.
   * **+**: Matches one or more occurrences of the preceding element.
   * **?**: Matches zero or one occurrence of the preceding element.
   * **\d**: Matches any digit (0-9).
   * **\w**: Matches any word character (alphanumeric + underscore).
   * **\s**: Matches any whitespace character (space, tab, newline).

Example pattern: **\d{3}-\d{2}-\d{4}** matches a social security number like **123-45-6789**.

**Anchors and Boundaries:**

1. **Anchors:**
   * **^**: Matches the start of a string.
   * **$**: Matches the end of a string.

Example: **^Start** matches a string that starts with "Start".

1. **Word Boundaries:**
   * **\b**: Matches a word boundary (position between a word character and a non-word character).

Example: **\bword\b** matches the word "word" as a whole word.

**Quantifiers:**

1. **Greedy and Lazy Quantifiers:**
   * **\*?**: Matches zero or more occurrences in a non-greedy way.
   * **+?**: Matches one or more occurrences in a non-greedy way.
   * **??**: Matches zero or one occurrence in a non-greedy way.

Example: **<.\*?>** matches the shortest possible XML tag.

**Grouping and Capturing:**

1. **Parentheses:**
   * **()**: Groups patterns and captures matched text.

Example: **(abc)+** matches "abc", "abcabc", etc.

**Escape Characters:**

1. **Escape Characters:**
   * **\**: Escapes a special character, allowing you to match it literally.

Example: **\\d+** matches one or more digits.

**Regex Methods in C#:**

1. **Regex.Match(string input, string pattern)**: Matches a single occurrence of the regular expression pattern in the input string.

csharpCopy code

Match match = Regex.Match(input, pattern);

1. **Regex.Matches(string input, string pattern)**: Matches all occurrences of the regular expression pattern in the input string.

csharpCopy code

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

1. **Regex.Replace(string input, string pattern, string replacement)**: Replaces all occurrences of the regular expression pattern with the specified replacement string.

csharpCopy code

string result = Regex.Replace(input, pattern, replacement);

1. **Regex.Split(string input, string pattern)**: Splits the input string into an array of substrings based on the occurrences of the regular expression pattern.

csharpCopy code

string[] substrings = Regex.Split(input, pattern);

**Example Usage:**

csharpCopy code

using System; using System.Text. RegularExpressions; class Program { static void Main() { string input = "Hello, my email is example@example.com and my phone number is 123-456-7890."; string emailPattern = @"\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b"; string phonePattern = @"\d{3}-\d{3}-\d{4}"; MatchCollection emails = Regex.Matches(input, emailPattern); MatchCollection phones = Regex.Matches(input, phonePattern); foreach (Match email in emails) { Console.WriteLine("Email found: " + email.Value); } foreach (Match phone in phones) { Console.WriteLine("Phone number found: " + phone.Value); } } }