Sintassi delle espressioni regolari

Un'espressione regolare rappresenta un criterio di testo costituito da caratteri ordinari, ad esempio le lettere dalla "a" alla "z", e da caratteri speciali, denominati *metacaratteri*. Il criterio descrive una o più stringhe di cui deve essere trovata una corrispondenza durante la ricerca all'interno del testo.

Esempi di espressioni regolari

Espressione	Corrispondenza
/^\s*$/	Corrisponde a una riga vuota.
/\d{2}-\d{5}/	Convalida un numero di ID costituito da 2 cifre, un trattino e altre 5 cifre.
/<\s*(\S+)(\s[^>]*)?>[\s\S]*<\s*\/\1\s*>/	Corrisponde a un tag HTML.

Nella tabella riportata di seguito viene fornito l'elenco completo dei metacaratteri e del relativo comportamento nel contesto delle espressioni regolari.

Carattere	Descrizione
\	Contrassegna il carattere successivo come carattere speciale, valore letterale, backreference o carattere di escape ottale. "n", ad esempio, corrisponde al carattere "n", mentre "\n" corrisponde a un carattere di nuova riga. La sequenza "\\" corrisponde a "\" e "\(" a "(".
^	Corrisponde alla posizione all'inizio della stringa di input. Se è stata impostata la proprietà **Multiline** dell'oggetto **RegExp**, ^ corrisponde inoltre alla posizione che segue "\n" o "\r".
$	Corrisponde alla posizione alla fine della stringa di input. Se è stata impostata la proprietà **Multiline** dell'oggetto **RegExp**, $ corrisponde inoltre alla posizione che precede "\n" o "\r".
*	Trova zero o più corrispondenze con la sottoespressione o il carattere precedente. "zo*", ad esempio, corrisponde a "z" e "zoo". * equivale a {0,}.
+	Trova una o più corrispondenze con la sottoespressione o il carattere precedente. "zo+", ad esempio, corrisponde a "zo" e "zoo", ma non a "z". + equivale a {1,}.
?	Trova zero o una corrispondenza con la sottoespressione o il carattere precedente. "do(es)?", ad esempio, corrisponde a "do" in "do" o a "does". ? equivale a {0,1}
{*n*}	*n* rappresenta un valore integer non negativo. Trova esattamente *n* corrispondenze. "o{2}", ad esempio, non corrisponde alla lettera "o" in "Bob", ma corrisponde alle due lettere "o" in "food".
{*n*,}	*n* rappresenta un valore integer non negativo. Trova almeno *n* corrispondenze. "o{2,}", ad esempio, non corrisponde alla lettera "o" in "Bob", mentre corrisponde a tutte le lettere "o" in "foooood". "o{1,}" equivale a "o+". "o{0,}" equivale a "o*".
{*n*,*m*}	*M* e *n* sono valori integer non negativi, in cui *n* <= *m*. Trova almeno *n* e al massimo *m* corrispondenze. "o{1,3}", ad esempio, corrisponde alle prime tre lettere "o" in "fooooood". "o{0,1}" equivale a "o?". Tra la virgola e i numeri non può essere inserito alcuno spazio.
?	Quando questo carattere si trova subito dopo uno degli altri quantificatori (*, +, ?, {*n*}, {*n*,}, {*n*,*m*}), il criterio di ricerca è specifico. Un criterio specifico trova il

	minor numero possibile di corrispondenze con la stringa con cui viene eseguita la ricerca, mentre il criterio generico predefinito trova il maggior numero possibile di corrispondenze con tale stringa. Nella stringa "oooo", ad esempio, "o+?" corrisponde a una singola lettera "o", mentre "o+" corrisponde a tutte le lettere "o".
.	Corrisponde a qualsiasi carattere singolo tranne "\n". Per ottenere una corrispondenza con qualsiasi carattere, incluso "\n", utilizzare un criterio come "[\s\S]".
x|*y*	Corrisponde a *x* o a *y*. Ad esempio, 'z|food' corrisponde a "z" o "food". '(z|f)ood' corrisponde a "zood" o "food".
[*xyz*]	Set di caratteri. Corrisponde a qualsiasi carattere incluso. "[abc]", ad esempio, corrisponde alla lettera "a" in "plain".
[^*xyz*]	Set di caratteri negativo. Corrisponde a qualsiasi carattere non incluso. "[^abc]", ad esempio, corrisponde alla lettera "p" in "plain".
[*a-z*]	Intervallo di caratteri. Corrisponde a qualsiasi carattere compreso nell'intervallo specificato. "[a-z]", ad esempio, corrisponde a qualsiasi carattere alfabetico minuscolo compreso nell'intervallo tra "a" e "z".
[^*a-z*]	Intervallo di caratteri negativo. Corrisponde a qualsiasi carattere non compreso nell'intervallo specificato. "[^a-z]", ad esempio, corrisponde a qualsiasi carattere non compreso nell'intervallo tra "a" e "z".
\b	Corrisponde a un inizio o a una fine di parola, ovvero alla posizione tra una parola e uno spazio. "er\b", ad esempio, corrisponde a "er" in "never" ma non a "er" in "verb".
\B	Corrisponde a caratteri che non costituiscono un inizio o una fine di parola. "er\B" corrisponde a "er" in "verb" ma non a "er" in "never".
\c*x*	Corrisponde a un carattere di controllo indicato da *x*. \cM, ad esempio, corrisponde a un carattere di ritorno a capo o a un caratere "M" di controllo. Il valore di *x* deve essere compreso nell'intervallo tra "A" e "Z" o tra "a" e "z". In caso contrario, c verrà considerato come valore letterale del carattere "c".
\d	Corrisponde a una cifra. Equivale a [0-9].
\D	Corrisponde a un carattere diverso da una cifra. Equivale a [^0-9].
\f	Corrisponde a un carattere di avanzamento modulo. Equivale a \x0c e \cL.
\n	Corrisponde a un carattere di nuova riga. Equivale a \x0a e \cJ.
\r	Corrisponde a un carattere di ritorno a capo. Equivale a \x0d e \cM.
\s	Corrisponde a qualsiasi carattere di spazio vuoto, come spazio, tabulazione, avanzamento modulo e così via. Equivale a [\f\n\r\t\v].
\S	Corrisponde a qualsiasi carattere diverso da uno spazio vuoto. Equivale a [^ \f\n\r\t\v].
\t	Corrisponde a un carattere di tabulazione. Equivale a \x09 e \cI.
\v	Corrisponde a un carattere di tabulazione verticale. Equivale a \x0b e \cK.
\w	Corrisponde a qualsiasi carattere alfanumerico, incluso il carattere di sottolineatura. Equivale a "[A-Za-z0-9_]".
\W	Corrisponde a qualsiasi carattere non alfabetico. Equivale a "[^A-Za-z0-9_]".
\x*n*	Corrisponde a *n*, dove *n* rappresenta un valore di escape esadecimale. I valori di escape esadecimali devono avere una lunghezza esatta di due cifre. "\x41", ad esempio, corrisponde ad "A". "\x041" equivale a "\x04" e "1". Consente l'utilizzo

	dei codici ASCII nelle espressioni regolari.
num	Corrisponde a *num*, dove *num* rappresenta un valore integer positivo. Costituisce un riferimento alle corrispondenze acquisite. "(.)\1", ad esempio, corrisponde a due caratteri identici consecutivi.
n	Identifica un valore di escape ottale o un backreference. Se *n* è preceduto da almeno *n* sottoespressioni acquisite, *n* costituisce un backreference. Se *n* è una cifra ottale (0-7), rappresenta un valore di escape ottale.
nm	Identifica un valore di escape ottale o un backreference. Se *nm* è preceduto da almeno *nm* sottoespressioni acquisite, *nm* costituisce un backreference. Se *nm* è preceduto da almeno *n* sottoespressioni acquisite, *n* costituisce un backreference seguito dal valore letterale *m*. Se non si verifica alcuna delle condizioni precedenti e *n* e *m* sono cifre ottali (0-7), *nm* corrisponde al valore di escape ottale *nm*.
\nml	Se *n* è una cifra ottale (0-3) e *m* e *l* sono cifre ottali (0-7), corrisponde al valore di escape ottale *nml*.
\u*n*	Corrisponde a *n*, dove *n* è un carattere Unicode espresso in quattro cifre esadecimali. \u00A9, ad esempio, corrisponde al simbolo di copyright (

esempio

Codice fiscale

```
^[a-zA-Z]{6}[0-9]{2}[abcdehlmprstABCDEHLMPRST]{1}[0-9]{2}([a-zA-Z]{1}

[0-9]{3})[a-zA-Z]{1}$

/*tra le tonde vi è l'identificativo del comune (codice catastale). Le lettere elencate
(abcd ecc.) indicano il mese di nascita mentre l'ultima lettera è quella di controllo.*/
```

Indirizzo posta elettronica

\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

\w un carattere alfabetico qualunque
+ uno o più volte il carattere che lo precede cioè in questo caso un carattere alfabetico qualunque
[-+.’] uno qualunque di questi caratteri
\w come sopra
+ uno o più volte il carattere che lo precede cioè in questo caso un carattere alfabetico qualunque
[-.] un carattere – o un carattere .
\w+ come gli altri
( )* quanto riportato in parentesi può essere ripatuto da 0 a N volte
\. Il carattere . da non confondere con . che corrisponde a qualsiasi carattere singolo tranne "\n"

Resto si ripete

Url internet

http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?

Metodo Regex.IsMatch (String)

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

public class Example
{
   public static void Main()
   {
      string[] partNumbers= { "1298-673-4192", "A08Z-931-468A",
                              "_A90-123-129X", "12345-KKA-1230",
                              "0919-2893-1256" };
      Regex rgx = new Regex(@"^[a-zA-Z0-9]\d{2}[a-zA-Z0-9](-\d{3}){2}[A-Za-z0-9]$");
      foreach (string partNumber in partNumbers)
         Console.WriteLine("{0} {1} a valid part number.",
                           partNumber,
                           rgx.IsMatch(partNumber) ? "is" : "is not");
   }
}
// The example displays the following output:
//       1298-673-4192 is a valid part number.
//       A08Z-931-468A is a valid part number.
//       _A90-123-129X is not a valid part number.
//       12345-KKA-1230 is not a valid part number.
//       0919-2893-1256 is not a valid part number.
```

L’espressione regolare è

```
^[A-Z0-9]\d{2}[A-Z0-9](-\d{3}){2}[A-Z0-9]$
```

Nella tabella seguente viene illustrato come viene interpretato il modello di espressione regolare.

Modello	Descrizione
^	Inizia la valutazione di corrispondenza all'inizio della stringa.
[A-Z0-9]	Ricerca la corrispondenza di un singolo carattere alfabetico compreso tra A e Z o di un qualsiasi carattere numerico.
\d{2}	Ricerca la corrispondenza di due caratteri numerici.
[A-Z0-9]	Ricerca la corrispondenza di un singolo carattere alfabetico compreso tra A e Z o di un qualsiasi carattere numerico.
-	Corrisponde a un trattino.
\d{3}	Ricerca l’esatta corrispondenza di tre caratteri numerici.

(-\d{3}){2}	Esegue la ricerca di un trattino seguito da tre caratteri numerici e ricerca la corrispondenza di due ricorrenze di tale modello.
[A-Z0-9]	Ricerca la corrispondenza di un singolo carattere alfabetico compreso tra A e Z o di un qualsiasi carattere numerico.
$	Termina la corrispondenza alla fine della stringa.

Metodo Regex.IsMatch (String, String)

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

public class Example
{
   public static void Main()
   {
      string[] partNumbers= { "1298-673-4192", "A08Z-931-468A",
                              "_A90-123-129X", "12345-KKA-1230",
                              "0919-2893-1256" };
      string pattern = @"^[a-zA-Z0-9]\d{2}[a-zA-Z0-9](-\d{3}){2}[A-Za-z0-9]$";
      foreach (string partNumber in partNumbers)
         Console.WriteLine("{0} {1} a valid part number.",
                           partNumber,
                           Regex.IsMatch(partNumber, pattern) ? "is" : "is not");
   }
}
// The example displays the following output:
//       1298-673-4192 is a valid part number.
//       A08Z-931-468A is a valid part number.
//       _A90-123-129X is not a valid part number.
//       12345-KKA-1230 is not a valid part number.
//       0919-2893-1256 is not a valid part number.
```

Metodo Regex.IsMatch (String, String, RegexOptions)

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

public class Example
{
   public static void Main()
   {
      string[] partNumbers= { "1298-673-4192", "A08Z-931-468a",
                              "_A90-123-129X", "12345-KKA-1230",
                              "0919-2893-1256" };
      string pattern = @"^[A-Z0-9]\d{2}[A-Z0-9](-\d{3}){2}[A-Z0-9]$";
      foreach (string partNumber in partNumbers)
```

```
         Console.WriteLine("{0} {1} a valid part number.",
                           partNumber,
                           Regex.IsMatch(partNumber, pattern, RegexOptions.IgnoreCase)
                                        ? "is" : "is not");
   }
}
// The example displays the following output:
//       1298-673-4192 is a valid part number.
//       A08Z-931-468a is a valid part number.
//       _A90-123-129X is not a valid part number.
//       12345-KKA-1230 is not a valid part number.
//       0919-2893-1256 is not a valid part number.
```

Metodo Regex.Match (String)

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

class Example
{
   static void Main()
   {
      string text = "One car red car blue car";
      string pat = @"(\w+)\s+(car)";

      // Instantiate the regular expression object.
      Regex r = new Regex(pat, RegexOptions.IgnoreCase);

      // Match the regular expression pattern against a text string.
      Match m = r.Match(text);
      int matchCount = 0;
      while (m.Success)
      {
         Console.WriteLine("Match"+ (++matchCount));
         for (int i = 1; i <= 2; i++)
         {
            Group g = m.Groups[i];
            Console.WriteLine("Group"+i+"='" + g + "'");
            CaptureCollection cc = g.Captures;
            for (int j = 0; j < cc.Count; j++)
            {
               Capture c = cc[j];
               System.Console.WriteLine("Capture"+j+"='" + c + "', position="+c.Index);
            }
         }
         m = m.NextMatch();
      }
   }
}
```

```
// This example displays the following output:
//       Match1
//       Group1='One'
//       Capture0='One', Position=0
//       Group2='car'
//       Capture0='car', Position=4
//       Match2
//       Group1='red'
//       Capture0='red', Position=8
//       Group2='car'
//       Capture0='car', Position=12
//       Match3
//       Group1='blue'
//       Capture0='blue', Position=16
//       Group2='car'
//       Capture0='car', Position=21
```

Il modello di espressione regolare (\w+)\s+(automobile) corrisponde alle occorrenze di una parola "automobile" con la parola che la precede. Viene interpretato come illustrato nella tabella seguente.

Modello	Descrizione
(\w+)	Trova la corrispondenza di uno o più caratteri alfanumerici. Equivale al primo gruppo di acquisizione.
\s+	Ricerca la corrispondenza di uno o più caratteri spazio vuoto.
(automobile)	Corrisponde alla stringa letterale "automobile." Equivale al secondo gruppo di acquisizione.

Metodo Regex.Match (String, String)

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

public class Example
{
   public static void Main()
   {
      string input = "ablaze beagle choral dozen elementary fanatic " +
                     "glaze hunger inept jazz kitchen lemon minus " +
                     "night optical pizza quiz restoration stamina " +
                     "train unrest vertical whiz xray yellow zealous";
      string pattern = @"\b\w*z+\w*\b";
      Match m = Regex.Match(input, pattern);
      while (m.Success) {
         Console.WriteLine("'{0}' found at position {1}", m.Value, m.Index);
         m = m.NextMatch();
      }
   }
}
// The example displays the following output:
//    'ablaze' found at position 0
//    'dozen' found at position 21
//    'glaze' found at position 46
//    'jazz' found at position 65
//    'pizza' found at position 104
//    'quiz' found at position 110
//    'whiz' found at position 157
//    'zealous' found at position 174
```

Il modello di espressione regolare \b\w*z+\w*\b viene interpretato come illustrato nella tabella seguente.

Modello	Descrizione
\b	Inizia la corrispondenza sul confine di parola.
\w*	Corrisponde a zero, uno o più, carattere alfanumerico.
z+	Corrisponde a una o più occorrenze del carattere z.
\w*	Corrisponde a zero, uno o più, carattere alfanumerico.
\b	Termina la corrispondenza sul confine di parola.

Metodo Regex.Match (String, String, RegexOptions)

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

public class Example
{
   public static void Main()
```

```
   {
      string pattern = @"\ba\w*\b";
      string input = "An extraordinary day dawns with each new day.";
      Match m = Regex.Match(input, pattern, RegexOptions.IgnoreCase);
      if (m.Success)
         Console.WriteLine("Found '{0}' at position {1}.", m.Value, m.Index);
   }
}
// The example displays the following output:
//        Found 'An' at position 0.
```

Il modello di espressione regolare \ba\w*\b viene interpretato come illustrato nella tabella seguente.

Modello	Descrizione
\b	Inizia la corrispondenza sul confine di parola.
a	Trova la corrispondenza del carattere "a".
\w*	Corrisponde a zero, uno o più, carattere alfanumerico.
\b	Termina la corrispondenza sul confine di parola.

Metodo Regex.Split (String)

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

public class Example
{
   public static void Main()
   {
      string pattern = @"\d+";
      Regex rgx = new Regex(pattern);
      string input = "123ABCDE456FGHIJKL789MNOPQ012";

      string[] result = rgx.Split(input);
      for (int ctr = 0; ctr < result.Length; ctr++) {
         Console.Write("'{0}'", result[ctr]);
         if (ctr < result.Length - 1)
            Console.Write(", ");
      }
      Console.WriteLine();
   }
}
// The example displays the following output:
//       '', 'ABCDE', 'FGHIJKL', 'MNOPQ', ''
```

Metodo Regex.Split (String, String)

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

public class Example
```

```
{
   public static void Main()
   {
      string input = "plum--pear";
      string pattern = "-";            // Split on hyphens

      string[] substrings = Regex.Split(input, pattern);
      foreach (string match in substrings)
      {
         Console.WriteLine("'{0}'", match);
      }
   }
}
// The method displays the following output:
//    'plum'
//    ''
//    'pear'
```

Metodo Regex.Split (String, String, RegexOptions)

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

public class Example
{
   public static void Main()
   {
      string pattern = "[a-z]+";
      string input = "Abc1234Def5678Ghi9012Jklm";
      string[] result = Regex.Split(input, pattern,
                                    RegexOptions.IgnoreCase);
      for (int ctr = 0; ctr < result.Length; ctr++) {
         Console.Write("'{0}'", result[ctr]);
         if (ctr < result.Length - 1)
            Console.Write(", ");
      }
      Console.WriteLine();
   }
}
// The example displays the following output:
//       '', '1234', '5678', '9012', ''
```

Metodo Regex.Replace (String, String)

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

public class Example
{
   public static void Main()
   {
      string input = "This is   text with   far  too   much   " +
                     "whitespace.";
      string pattern = "\\s+";
      string replacement = " ";
      Regex rgx = new Regex(pattern);
      string result = rgx.Replace(input, replacement);

      Console.WriteLine("Original String: {0}", input);
      Console.WriteLine("Replacement String: {0}", result);
   }
}
// The example displays the following output:
//       Original String: This is   text with   far  too   much   whitespace.
//       Replacement String: This is text with far too much whitespace.
```