Espressioni regolari in PERL

Espressione regolare: definisce la grammatica di un minilinguaggio ed è fondamentalmente una sintassi mediante la quale rappresentare un insieme S di stringhe.

Espressione regolare in PERL: è un "qualcosa" che di solito viene racchiuso tra / e /:

```
/<regexp>/
```

Operatori di confronto

Operatore di ricerca (m: matching)

```
$string =~ m/<regexp>/;
```

L'interprete dell'espressione regolare confronta la sintassi di <regexp> con la stringa contenuta in $string e se almeno una parte della stringa è riconosciuta come espressione del linguaggio di <regexp> allora restituisce VERO in un contesto scalare.

```
if($string =~ m/<regexp>/){
     print "VERO\n";
}
```

Operatore di sostituzione (s: substitution)

```
$string =~ s/<regexp>/<subst_string>/;
```

L'interprete dell'espressione regolare confronta la sintassi di <regexp> con la stringa contenuta in $string e sostituisce la parte della stringa riconosciuta come espressione del linguaggio di <regexp> e la sostituisce con <subst_string>.

Alfabeto

- Letterali: a, b, c, …, A, B, C, …, 1, 2, 3, …
  `/Mario/` ➔ S={"Mario"} e non S={"Mario","mario"}

- Metacaratteri: \ | ( ) [ ] { } ^ $ * + ? .

Il carattere backslash \ trasforma:
- un metacarattere in un letterale ➔ \(è il letterale parentesi tonda e \\ è il letterale backslash
- un letterale in un metacarattere ➔ \t rappresenta una tabulazione.

Sintassi

Carattere .

Il . rappresenta qualsiasi carattere eccetto il newline.

Quantificatori

I quantificatori indicano quante volte l'elemento che li precede in <regexp> deve essere riscontrato e sono:
- * ➔ 0 o più volte
- + ➔ 1 o più volte
- ? ➔ 0 o 1 volta
- {m,n} ➔ almeno m volte ma non più di n volte
- {n,} ➔ almeno n volte
- {n} ➔ esattamente n volte

esempi:
`/a+b*/` ➔ S={"a","aa","aaaa…", "ab","abb","aabb","aaa…bbbb…"}, ma "b","bb","bbb…" non appartengono a S

`/a{2,5}/` ➔ S={"aa","aaa","aaaa","aaaaa"}

Alternative

Per evidenziare alternative su usa |

esempio: `/Mario|Luca/` ➔ S{"Mario","Luca"}

Raggruppamenti

Per raggruppare gli elementi di un'espressione regolare si usano le parentesi tonde ()

esempi:
`/(Mario|Luca) Rossi/` ➔ S={"Mario Rossi","Luca Rossi"}

`/(casa){5}/` ➔ S={"casacasacasacasacasa"}
`/casa{5}/` ➔ S{"casaaaaa"}

Classe di caratteri

Una classe di caratteri è una lista di caratteri tra parentesi quadre [] e rappresenta un carattere qualsiasi della lista.

esempio: `/[abcd]1/` ➔ S={"a1", "b1", "c1, "d1"}

Il metacarattere ^ indica il complemento di una classe.

esempio: `/[^abcd]1/` ➔ S={"e1", "f1", "g1", …, "z1"}

Intervallo

Un intervallo di caratteri è indicato con - : a-z, A-Z, 0-9

esempio: `/[a-z]/` ➔ S={“a”, “b”, …, “z”}

Il segno meno è \-

Caratteri speciali

- \n ➔ nuova riga
- \r ➔ ritorno a capo
- \t ➔ tabulazione
- \f ➔ formfeed
- \d ➔ cifra in 0-9 ([0-9])
- \D ➔ non-cifra in 0-9 ([^0-9])
- \w ➔ carattere di parola ([a-zA-Z0-9_])
- \W ➔ carattere non di parola ([^a-zA-Z0-9])
- \s ➔ spazio vuoto ([\t\n\r\f])
- \S ➔ carattere non spazio vuoto ([^ \t\n\r\f])

Asserzioni e atomi

Gli elementi di un’espressione regolare possono essere:
- asserzioni: elementi di lunghezza zero (esempio: confine di parola)
- atomi: elementi di lunghezza diversa da zero (esempio: un singolo carattere a cui segue un quantificatore)
- nessuno dei due (esempio: | per le alternative)

Asserzioni

- \b ➔ confine di parola (tra \w e \W)
- \B ➔ tutto tranne confine di parola
- ^ ➔ inizio della riga o della stringa (\n viene ignorato)
- $ ➔ fine della riga o della stringa (\n viene ignorato)
- \A ➔ inizio della stringa (\n viene ignorato)
- \Z ➔ fine della stringa (\n viene ignorato)
- (?=expr) ➔ asserzione di riscontro di estensione nulla; presenza successivamente di una sequenza del linguaggio expr (non è un raggruppamento)
- (?!expr) ➔ asserzione di non riscontro di estensione nulla; assenza successivamente di una sequenza del linguaggio expr (non è un raggruppamento)

esempio di confine di parola

`$string =~ m/a234bg\b/` è VERA ad esempio se $string=”a234bg ” ma non se $string=”a234bgg”

esempi di inizio e fine della stringa e della riga:

```
$string=”questa stringa\nrisulta essere\nsu tre
righe\n”;
```

`$string =~ m/\Aq/` è VERA
`$string =~ m/e\Z/` è VERA
`$string =~ m/^q/` è VERA
`$string =~ m/e$/` è VERA

`$string =~ m/^r/` è VERA se $string viene pensata su più righe, altrimenti è FALSA
`$string =~ m/a$/` è VERA se $string viene pensata su più righe, altrimenti è FALSA
`$string =~ m/\Ar/` è sempre FALSA
`$string =~ m/a\Z/` è FALSA

esempio di (?=expr) e di (?!expr)

```
$string1="mario\tluca";
$string2="mario-luca";
```

`$string1 =~ m/mario(?=\t)/` è VERA
`$string2 =~ m/mario(?=\t)/` è FALSA
`$string2 =~ m/mario(?!\t)/` è VERA

Nel caso di $string1 la sequenza riscontrata è "mario" e non "mario\t".

```
$string1 =~ m/(?!\t)mario/;
```

non significa cercare un riscontro di "mario" preceduto da tabulazione.
Corretto sarebbe:

```
$string1 =~ m/(?!\t).mario/);
```

Alcune regole di confronto…

1) Viene cercata la posizione più a sinistra in cui si riscontra l'intera espressione regolare, senza andare oltre nell'esplorazione di altri riscontri. La stringa viene esaminata da sinistra a destra finché non viene trovato un riscontro di regexp o finché il confronto fallisce. La sequenza riscontrata deve arrivare alla fine dell'espressione regolare senza raggiungere la fine della stringa, a meno che non vi sia un'asserzione di fine stringa \Z o $.

```
$string="mario";
```

`$string =~ m/ar/;` è VERA
`$string =~ m/ar$/;` è FALSA

ATTENZIONE: `$string =~ m/a*/;` trova sempre riscontro all'inizio della stringa in quanto la stringa "" è parte del linguaggio dell'espressione regolare

2) L'espressione regolare è intesa come un insieme di alternative separate dal carattere |. Se non esiste | allora l'alternativa è una sola. L'espressione regolare trova riscontro in una stringa se una delle sue alternative trova riscontro nella stringa.

```
$string =~ m/Mario|Luca/;
```

3) Un'alternativa viene soddisfatta se ogni suo elemento trova riscontro sequenzialmente. La stringa è esaminata da sinistra a destra. Se l'alternativa non viene soddisfatta si passa alla successiva nell'espressione regolare.

4) Il meccanismo dei quantificatori è "ingordo". Prima infatti si esamina la sequenza con il numero massimo possibile di atomi riscontrati. Se questa sequenza non soddisfa l'espressione regolare allora diminuisce di 1 il numero di atomi fino al

minimo consentito finché non riesce a trovare il riscontro dell'espressione regolare.

```
$string="aaaabbbaaaaabbbaaaa";
$string =~ m/.*bbb/;
```

la sequenza riscontrata è "aaaabbbaaaaabbb"

Se si vuole che la sequenza riscontrata sia "aaaabbb" si deve scrivere

```
$string =~ m/.*?bbb/;
```

In questo modo con ? dopo il quantificatore * si parte dall'esame della sequenza con il numero minimo di atomi di tipo punto (.), che sarà quindi 0.

Backreference

I numeri interi preceduti da \ forniscono un modo per fare riferimento ai raggruppamenti in parentesi tonde. Più precisamente \1 \2 etc. equivalgono alle sequenze effettivamente riscontrate (e non all'espressione regolare tra ()) dagli elementi nelle parentesi tonde contando da sinistra a destra. Nel caso le () siano seguite da un quantificatore allora il riferimento è relativo all'ultima delle sequenze riscontrate in successione.

```
$string1="a1234 a1234";
$string2="a1234 a124";
$string1 =~ m/(\w+)\s\1/; è VERA
$string2 =~ m/(\w+)\s\1/;  è FALSA
$string2 =~ m/(\w+)\s\w+/;  è VERA
```

Le variabili $1, $2, etc. funzionano all'esterno di un criterio di ricerca e contengono le sequenze effettivamente riscontrate nei vari raggruppamenti in () dell'espressione regolare e contate da sinistra a destra.

La variabile $& restituisce l'intera sequenza riscontrata

```
$string="a1234 a1234";
if($string =~ m/(\w+)\s\1/){
     print $1, "\n";
     print $&, "\n";
}
```

stampa "a1234\n" e "a1234 a1234\n".

Le variabili $ e $' restituiscono rispettivamente ciò che precede e ciò che segue $&.

Nel caso delle asserzioni (?=expr) e (?!expr), le stringhe riscontrate non fanno parte di $&.

Operatori di confronto

Operatore **m**:

```
$string =~ m/<regexp>/;
```

- restituisce un valore booleano in un contesto scalare ("1" per VERO e "" per FALSO)

Esempio di contesto scalare ➔

```
if($string =~ m/<regexp>/){
     print "VERO\n";
```

```
}
$b=($string =~ m/<regexp>/);
print $b, "\n";
```

- restituisce la lista delle sequenze riscontrate dalle eventuali parentesi tonde in un contesto di lista (lista vuota se il confronto fallisce). Se non ci sono parentesi restituisce una lista di un solo elemento ("1") se il riscontro è positivo, altrimenti restituisce la lista vuota

Esempio di contesto di lista ➔

```
@list = ($string =~ m/<regexp>/);
```

Modificatori:

➔ g

```
$string =~ m/<regexp>/g;
```

restituisce tutti i riscontri possibili nella stringa. In un *contesto di lista* restituisce la lista di tutti i riscontri legati a tutti i raggruppamenti in parentesi tonde. Se non esistono parentesi tonde, restituisce la lista di tutti i riscontri dell'espressione. In un *contesto scalare*, cerca un nuovo riscontro ogni volta che viene invocato fino a che i riscontri non si esauriscono

➔ i

```
$string =~ m/<regexp>/i;
```

non fa distinzione tra maiuscole e minuscole

➔ m

```
$string =~ m/<regexp>/m;
```

tratta la stringa come composta da più righe

➔ s

```
$string =~ m/<regexp>/s;
```

tratta la stringa come una singola riga e \n è incluso nell'insieme di caratteri rappresentati dal punto (.)

`$string !~ m/<regexp>/;`

restituisce, in un contesto scalare, VERO se <regexp> NON è riscontrata in $string, altrimenti FALSO

Operatore **s**:

`$string =~ s/<regexp>/<subst_string>/;`

restituisce il numero delle sostituzioni effettuate (più di una se si usa il modificatore g). <subst_string> è da pensare come una stringa tra doppi apici (“) e NON come un’espressione regolare.

Modificatori:

➔ g

```
$string =~ s/<regexp>/<subst_string>/g;
```

sostituisce TUTTI i riscontri di <regexp> in $string con la stringa <subst_string>

➔ i

```
$string =~ s/<regexp>/<subst_string>/i;
```

non fa distinzione tra maiuscole e minuscole

➔ m

```
$string =~ s/<regexp>/<subst_string>/m;
```

tratta la stringa come composta da più righe

➔ s

```
$string =~ s/<regexp>/<subst_string>/s;
```

tratta la stringa come una singola riga e \n è incluso nell’insieme di caratteri rappresentati dal punto (.)

➔ e

```
$string =~ s/<regexp>/<subst_string>/e;
```

tratta <subst_string> come un’espressione qualsiasi e non come una stringa tra doppi apici (“).

Esempio:

```
$string=”abc123abc”;
$string =~ s/(\d+)/$1*2/e;
```

Dopo la sostituzione in $string è contenuta la stringa “abc246abc”

Ulteriori esempi…

Esempio 1

```
$string=”questa stringa\nrisulta essere\nsu tre
righe\n”;
```

```
$b1=($string =~ m/\Aq/);
if($b1){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b2=($string =~ m/e\Z/);
if($b2){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b3=($string =~ m/^q/);
if($b3){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b4=($string =~ m/e$/);
if($b4){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b5=($string =~ m/\Ar/);
if($b5){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b6=($string =~ m/^r/);
if($b6){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b7=($string =~ m/a\Z/);
if($b7){
     print “VERO\n”;
```

```
}
else{
     print “FALSO\n”;
}

$b8=($string =~ m/a$/);
if($b8){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b9=($string =~ m/\Ar/m);
if($b9){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b10=($string =~ m/^r/m);
if($b10){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b11=($string =~ m/a\Z/m);
if($b11){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$b12=($string =~ m/a$/m);
if($b12){
     print “VERO\n”;
}
else{
     print “FALSO\n”;
}

$string =~ m/^(.+)/;
print $1;
$string =~ m/^(.+)/s;
print $1;
```

OUTPUT:

VERO
VERO
VERO
VERO
FALSO
FALSO
FALSO
FALSO
FALSO
VERO
FALSO
VERO
questa stringaquesta stringa
risulta essere
su tre righe

Esempio 2

```
$string="***a1234 b345***a567 b789***";
$b=($string =~ m/(\w+)\s(\w+)/);
print "Il confronto è ",
if($b){
     print "positivo\n";
}
else{
     print "negativo\n";
}
```

OUTPUT:

Il confronto è positivo

Esempio 3

```
$string="***a1234 b345***a567 b789***";

@list=($string =~ m/(\w+)\s(\w+)/);
print "Prima parola: ", $1, "\n";
print "Seconda parola: ", $2, "\n";
for($i=0; $i<=$#list; $i++){
     print "La parola è $list[$i]\n";
}

print "xxxxxxxxxx\n";

@list=($string =~ m/\w+\s\w+/);
for($i=0; $i<=$#list; $i++){
     print "Il riscontro è $list[$i]\n";
}
```

OUTPUT:

Prima parola: a1234
Seconda parola: b345
La parola è a1234
La parola è b345
xxxxxxxxxx
Il riscontro è 1

Esempio 4

```
$string="***a1234 b345***a567 b789***";

@list=($string =~ m/(\w+)\s(\w+)/g);

for($i=0; $i<=$#list; $i++){
     print "La parola è $list[$i]\n";
}

print "xxxxxxxxxx\n";

@list=($string =~ m/\w+\s\w+/g);
for($i=0; $i<=$#list; $i++){
     print "Il riscontro è $list[$i]\n";
}
```

OUTPUT:

La parola è a1234
La parola è b345
La parola è a567
La parola è b789
xxxxxxxxxx
Il riscontro è a1234 b345
Il riscontro è a567 b789

Esempio 5

```
$string="***a1234 a1234***a567 b789***";

@list=($string =~ m/(\w+)\s\1/g);

for($i=0; $i<=$#list; $i++){
     print "La parola è $list[$i]\n";
}
```

OUTPUT:

La parola è a1234

Esempio 6

```
$string="***a1234 b345***a567 b789***";
```

```
while($string =~ m/(\w+)\s(\w+)/g){
     print "Il riscontro è $&\n";
     print "Prima parola: $1\n";
}
```

OUTPUT:

Il riscontro è a1234 b345
Prima parola: a1234
Il riscontro è a567 b789
Prima parola: a567